

【書類名】 明細書

【発明の名称】 ソフトウェア生成方法

【技術分野】

【0001】

本発明は、数学的考え方にたったソフトウェア生成方法に係り、特に、隣接行列、トポロジカル・ソートを用いたソフトウェア生成方法に関する。

【背景技術】

【0002】

(1) ソフトウェア開発の課題

高い品質を持つソフトウェアを容易に迅速に製造することは、ソフトウェア開発研究分野の基本的な関心事である。ソフトウェア開発の生産性、保守性、品質を向上するための種々の方法論および技術が考案され提案されてきたが、いまだにこれらの課題を本質的に解決できたものはない。その理由の1つは、ソフトウェア自身が複雑なものであり、捕らえにくいものであるからであり、もう1つの理由は、現在の方法論に限界があるからである。実際、提案されたほとんどすべての方法論は、明快に理解でき、修正可能なシステムの製造に失敗しているのみならず、それらは、非常に広範囲の能力、技術および知識を持つ専門家にしか利用できないものといまだに見なされている。そのため、人件費や維持費が高いものになり、ソフトウェア上で実行させるためには広範なチェックが必要になる。従来のソフトウェア開発手法の概要と課題をまとめると以下のようになる。

(2) 手続き型プログラミング

一般に、手続き型プログラミングは、その記述言語の扱いやすさから、業務システム開発とその実装において効果を発してきた。しかし、いまだ要件を正確に実装することの困難性、生産性や保守性の向上という重大な課題が残されている。

(3) 宣言型プログラミング

一方、要件定義そのものである宣言によってプログラミングを行う宣言型のプログラミングは、人工知能、エキスパートシステムをはじめ論理的証明 (logical reasoning)、推論 (inference)、ユーザ情報収集 (user intelligence gathering) など、非計算系のアプリケーション分野で使用されている。宣言型プログラミングは、制御手順でなく、

要件定義されているデータ間の関係を宣言し、その宣言に基づいてプログラミングを行うので、そのメリットは、制御手順などの手続きを明示的に指定する必要がある手続き型プログラミングと異なり、要件を正確に実装できるということである。

【0003】

しかし、宣言型プログラミングにも、実行制御の構造上、実行時の効率が良くないという問題がある。また、宣言型プログラミングを行うための言語は使いこなすのが難しいという欠点がある。宣言型プログラミングを行うための言語には、論理式を用いる論理型言語（たとえばPROLOG）や対象の状態を数学的な関数として記述する関数型言語（たとえばLISP）がある。宣言しただけでは値は生成されないから、値を生成するためには、宣言型言語は、宣言を実行するための手続きの論理メカニズムを持っている。これらの手続きの論理メカニズムには、あらゆる宣言を実行できるようにするために多くの条件があり、数学や論理学の知識も必要で、使いこなすのが難しい。このために、プログラミング言語として普及していない。人間の自然言語に近い構造を持つ手続き型言語による、手続き型プログラミングの方が普及しているゆえんである。

（4）オブジェクト指向プログラミング

データとそれを操作する手続き（メソッドと呼ぶ）をオブジェクトと呼ばれるひとまとまりの単位として一体化し、オブジェクトの組み合わせとしてプログラムを記述するプログラミング技法として、オブジェクト指向プログラミングがある。オブジェクトという独立性の高いモジュールによってプログラムが構成されるため、変更・修正の影響の範囲が限定され、またオブジェクトの単位で再利用がしやすくなるなどのメリットがある。

【0004】

しかし、要件を、どのような単位のオブジェクトによって実装するのか、さらに、構成要素をどのように組み立てるのかについては厳格な規則はなく、また、構造化設計アプローチのような実行順序についての規制もない。実際の開発の場においては、オブジェクト間の関係が完全に一貫性を持って適用されず、多くの場合、開発成果物に大量のオブジェクトが生成されていても、作成者自身を除けば誰もそれを理解していないということになる。すなわち、完成したソフトウェアは規則性のない機能のかたまりとなり、最終的に解体したり、再使用したりするのが困難になる傾向が強い。

【特許文献1】

国際公開第97/16784号パンフレット

【特許文献2】

国際公開第98/19232号パンフレット

【特許文献3】

国際公開第 99 / 49387 号パンフレット

【特許文献 4】

国際公開第 00 / 79385 号パンフレット

【特許文献 5】

国際公開第 02 / 42904 号パンフレット

【特許文献 6】

国際公開第 2004 / 68342 号パンフレット

【特許文献 7】

特開 2002-202883 号公報

【非特許文献 1】

グリーン・ウィンスケル著、「プログラミング言語の公式意味論」、MIT プレス、1993 年

【非特許文献 2】

ハンネ・リース・ニールソン、フレミング・ニールソン著、「アプリケーションについての 意味論 公式紹介」、ジョン・ワイリー・アンド・サンズ、1992 年

【非特許文献 3】

根来 文生著、「Lyee ソフトウェアの原理」、21 世紀（IS2000）における情報社会についての国際会議会報、日本、会津、2000 年 11 月 5 日-8 日、pp441-446

【非特許文献 4】

根来 文生著、「ソースコード生成についての高密度処理方法」、システミクス、サイバネティクス及びインフォマティクスについての第 5 回世界多極会議会報、（SCI2001）、米国、オーランド、2001 年 7 月 22 日-25 日

【発明の開示】

【発明が解決しようとする課題】

【0005】

(1) Lyee 開発方法論

最近、Lyee (governmental methodology for sof

t w a r E p r o v i d e n c Eの語尾をとったもの。「リー」と読む。発明者は根来文生。) と呼ばれる新しく、非常に有望な方法論が提案された。Lyeeは、要件からソフトウェアを自動的に開発する新しい方法である。

【0006】

Lyeeの場合、宣言型でありながら、宣言を実行するための特別な論理メカニズムを必要とせず、宣言の実行は、手続き型プログラミングと同じく、単純な値の代入という手続きである。従って、使用する言語は一般に普及している手続き型言語でよい。つまり、Lyeeは要件を正確に実装できるという宣言型の利点と、記述が容易な手続き型言語を使用できるという手続き型の利点をあわせ持っているのである。

【0007】

Lyee開発方法の概要については、本願の出願人による特願2004-272400および特許文献6を引用し、これを開示の一部となす。以下に概要を述べる。

(2) Lyeeにおける宣言とは

Lyeeにおいては、宣言する抽象化の対象は変数である。宣言された変数を、Lyeeでは単語と呼ぶ。一般的に、仕様が宣言的であるとは、宣言される抽象化の単位は記述の順序性が排除されることと、等式として表現され右辺から左辺、左辺から右辺への双方向性があることである。Lyeeでは、単語は記述の順序性が排除されるが、双方向性をもたず、このことによって、枠組みが簡潔になっている。

【0008】

単語の宣言の主な要素は、単語名、定義式、定義式の計算条件、入出力属性、値の属性などである。定義式とは、他の単語との関係を示す関係式である。たとえば、単語aは定義式 $a=b+c$ のように宣言できる。そして、この宣言の実行は、関係式を代入式として実行するだけである。たとえば、単語aの宣言 $a=b+c$ を実行するには(aの値を求めるには)、右辺の変数bおよびcに値を代入すればよい。

(3) 手続き型の枠組みで宣言を実行するメカニズム

手続き型言語の枠組みで、実装していながら、宣言の記述の順序性排除を実現しているのは、Lyeeが提供する定型構造のプログラム構造である。プログラムを処理するコンピュータが、現在使われている、命令をシーケンシャルに処理するノイマン型である限り、手続き型言語であれば、なんらかの方法で順序性の問題を解決しなければならない。他の宣言型プログラミングは、宣言型言語がそれを解決している。手続き型言語を用いるLyeeは、プログラム構造によって解決している。

【0009】

そのプログラム構造は、単語単位の宣言を実行する定型構造のモジュール(以下、「宣

言実行モジュール」と呼ぶ。) からなり、それらのモジュールは、パレットと呼ぶ単位で集合化されている。パレット内の、宣言実行モジュールの実行順序は、いかなる配列でもよい。パレット内の実行制御モジュールが、パレット内の全ての宣言実行モジュールの実行が完了するまで(すなわち、値が生成できまるで) 宣言実行モジュールの実行を繰り返すよう実行を制御する。

【0010】

パレットは、入力単語の宣言実行のためのW02パレット、出力単語の宣言のうち計算条件を実行するためのW03パレット、出力単語の宣言のうち定義式実行と出力を実行するためのW04パレットがある。これらの3種類のパレットが1組となって、基本構造という集合単位を成している。すなわち、Lyee構造のプログラムは、基本構造単位に分割され、その基本構造は3種類のパレットに分割され、パレットは宣言実行モジュールに分割される、という構造をとっているのである。基本構造は、出力論理体(すなわち同時に同じ媒体に出力される単語のグループ)の単位につくられる。Lyeeプログラムのこの分割構造を図示したのが、処理経路図(PRD: Process Route Diagram)である。

【0011】

Lyeeによるプログラム全体は、単語単位の宣言を実行する定型構造の宣言実行モジュールによって構成され、しかもそれら宣言実行モジュールでプログラム全体を構成する方法も定型化されている。従って、この定型構造の宣言実行モジュールに、単語単位の要件の宣言を代入すれば、自動的にプログラムを生成することも可能である。

【0012】

実際、Lyeeによるプログラムの自動生成は、「LyeeALL」と呼ばれるツールによって実現されている。LyeeALLは、いわば、定型構造の宣言実行モジュールのコード(「テンプレート」と呼ぶ。テンプレートはひな形の意。)の所定の空欄に、要件の宣言を自動的に代入し、それらの宣言実行モジュールを所定の方法に従って構成して1つのプログラムを構築するツールである。

【0013】

Lyee方法論によるソフトウェア開発では、技術者は2つのグループに分かれ、それぞれ特化した2種類の仕事を行うことになる。1つのチームは、OS、ミドルウェア、データベースなど、Lyee構造プログラムが動く外部環境に係わる技術に習熟し、Lyee構造プログラムのテンプレートに、外部環境ごとのパラメータを設定する。もう1つのチームは、ユーザ要件定義を行い、要件をLyee構造プログラムのテンプレートに実装するために、Lyee方法論に従った要件の宣言を行う。このような分業制による開発が可能になるため、Lyee方法論の開発においては、プログラマ全員が広範な外部環境に係わる技術を持つ必要がなく、そのようなスキルを持つ人材不足の問題も解消できる。

(4) 従来のLyee構造の課題

上述のように、要件の正確な実装、コード生成の自動化、保守の容易性など、大きなメリットがあるLyee方法論であるが、その繰り返しと分割の構造には課題もある。繰り返し処理のために、実行速度が遅く、分割構造のためにプログラムサイズが大きい、とされている。

【0014】

一方、Lyeeでは、基本構造と呼ぶ単位に、要件を分割して仕様を捉える。画面の遷移、データベースのキーの関係などの実行順序に関する明白な情報を単語のグループ(record)を単位として捉え、そのグループに属す単語の実行条件をまとめて把握する。現状のLyee構造は、この分割構造を、そのままコード化しているため、プログラムサイズが大きく、実行速度も遅くなっている。

【0015】

Lyee構造を、その宣言的プログラミングの利点をそのままに、要件の意味を変更せずに維持しながら、不要な反復がなく、よりサイズが小さく、効率的処理速度のプログラムに改善することは重要な課題である。

【0016】

不要な繰り返しをなくして、実行速度の改善をはかる方法としては、トポロジカル・ソートによって、宣言実行モジュールの配置順序を自動的に最適化する(1巡で宣言実行が完了する順序化)方法がすでに提案されている(PCT/JPO3/09591)。パレット内のモジュールにトポロジカル・ソートを行って、パレット内の無駄な繰り返しを排除することができる。

【0017】

しかし、Lyeeの分割構造の枠組みをそのままにしてトポロジカル・ソートを行っただけでは、その効果には限界がある。なぜなら、パレット間、基本構造間にも繰り返しがあり、パレット内のトポロジカル・ソートによる最適順序化により、繰り返し回数は最小限になるもの、全ての値の生成が1巡では完了しないからである。また、分割構造そのものが、プログラムサイズを大きくしている一因である。

【0018】

従って、繰り返しと分割の構造を持ったLyeeプログラムのサイズと処理速度の問題を、さらに大きく改善するためには、パレットおよび基本構造という分割単位をなくし、1本のプログラムとして実装する要件の範囲の宣言を実行するモジュールに対してトポロジカル・ソートを行うことが有効と思われる。

【0019】

本発明は、従来のLyee構造プログラムの分割単位をなくし、プログラム全体に対してトポロジカル・ソートを行う方法に関するものである。これにより、Lyee方法論によるソフトウェアの実行速度の向上、プログラムサイズの改善を実現することを目的とする。

【課題を解決するための手段】

【0020】

かかる課題を解決するために本発明は、1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から、要件充足に必要な全ての、L2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）のいずれかの宣言の実行単位を規定する第1のステップと、

前記に規定された、全てのL2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）の（半）順序関係を定義する第2のステップと、

前記ステップによって定義された（半）順序関係で規定される前記L2処理、L処理、I2処理、O4処理に対してトポロジカル・ソートを行う第3のステップと、

前記第3のステップによって並び替えられた宣言の実行単位の順序列にしたがって該宣言実行単位に該当するLyee方法論に基づく所与のコード列を配置する第4のステップと

を具備する。

【0021】

かかる構成を有する本願発明によれば、第1のステップによって規定された単語単位の宣言を、L2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）、の単位で、最短順序で実行するプログラムが作成されることとなり、Lyee方法論を用いたソフトウェア生成がより効率化する。つまり、これらの各ステップの機能は総て自動化できるので、結局繰返し処理の排除→ソフトウェア生成まで一貫して自動処理が可能となり、ソフトウェア生産速度、効率は各段に向上する。

【0022】

ここで、「全てのL2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）の（半）順序関係を定義する」とは、宣言で用いられる単語に着目して相互の順序関係をたとえば有向グラフで表すことをいう。順序関係としては半順序及び全順序が考えられる。「有向

グラフ」とは、ソフトウェア開発要望者であるユーザの発する要件に含まれるそれぞれの宣言実行単位をノードによって表し、 $a \leq b$ (aからbが作られることを意味する記号として「 \leq 」を定義する。以下同じ。)という始点と終点の間の半順序関係 (または全順序関係) を矢印によって表したものをいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0023】

また、「トポロジカル・ソート」とは、上記で得られた有向グラフに対して (後述の) 深さ優先探索を行い、行きついたところから探索経路を戻るときにノードを取得してゆくことでノード列 (数列) を整列する機能をいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。このトポロジカル・ソートに際しては、上記で得られた有向グラフを隣接行列で表し、これに深さ優先探索を行い、零冪行列を生成するように動作させてもよい。

【0024】

ここで、「深さ優先探索」とは、

- (1) 始点であるノードを出発してノード間の連結がとれているノードを訪問が重複しないように進み、
- (2) 行く場所がなくなったら行く場所のあるノードまで戻り (再帰)、
- (3) 戻ったノードから再び (1) の要領で進み、
- (4) 進むところがなくなったら終了する、

というアルゴリズムをいう。

【0025】

「探索」とは、上記の深さ優先探索 (Depth-First Search) のほか、幅優先探索 (Breadth-First Search)、反復深化法 (Iterative Deeping)、発見的探索 (Heuristic Search)、ヒル・クライミング (Hill Climbing)、最適優先探索 (Best-First Search)、オイラー (Euler) の一筆書き、ダイクストラ (Dijkstra) 法等を含む各種アルゴリズムをいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0026】

また、ここで用いる「ソフトウェア」とは広義の意である。即ち、ユーザからの開発

要望及び当該要望にあった目的プログラムの双方を包含する概念である。

【0027】

上記のような構成を備えることにより、本願発明では、繰返しを回避するので、より高効率、高速、高パフォーマンスを備えたソフトウェア開発を実現することが可能となる。さらに、繰返しをなくす前処理を自動で行い、こうして得られた前処理済の宣言実行単位のプログラムをもとにL y e e（登録商標）方法論により自動的に目的プログラム生成を行う。つまり、ユーザ要件の洗練化（整列）から目的プログラムの生成に至るまで自動化することが可能となる。これにより、ソフトウェア生産の大幅な効率向上、生産性向上、品質向上等、ソフトウェア産業上に大きな効果をもたらす。

【0028】

本発明の異なる実施体としての「開発対象のソフトウェア」を生産するためのプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、ソフトウェア開発管理装置は、1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から、要件充足に必要な全ての、L 2 処理（入力単語の属性チェック処理）、L 処理（出力単語の値生成処理）、I 2 処理（論理体入力処理）、O 4 処理（論理体出力処理）のいずれかの宣言の実行単位を規定する手段と、前記に既定された、全てのL 2 処理（入力単語の属性チェック処理）、L 処理（出力単語の値生成処理）、I 2 処理（論理体入力処理）、O 4 処理（論理体出力処理）の（半）順序関係を定義する手段と、前記ステップによって定義された（半）順序関係で規定される前記L 2 処理、L 処理、I 2 処理、O 4 処理に対してトポロジカル・ソートを行う手段と、前記第3のステップによって並び替えられた宣言の実行単位の順序列にしたがって該宣言実行単位に該当するL y e e 方法論に基づく所与のコード列を配置する手段とを具備するように構成することもできる。

【0029】

本発明はさらに、上述の「開発対象のソフトウェアを生産する方法」によって生産されたソフトウェア、及び当該ソフトウェアが搭載された記録媒体或いは当該ソフトウェアが搭載された装置（ハードウェア）としても実現されるが、この場合の本発明は、1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から規定された、要件充足に必要な全ての、L 2 処理（入力単語の属性チェック処理）、L 処理（出力単語の値生成処理）、I 2 処理（論理体入力処理）、O 4 処理（論理体出力処理）の宣言の実行単位が、単語単位に宣言された要件から定義した（半）順序関係に基づいて行われるトポロジカル・ソートによって並び替えられた順序列にしたがって、該当するL y e e 方法論に基づく所与のコード列を配置する手段とを具備するように構成することもできる。

ード列として構成されることもできる。

【0030】

またさらに本発明は、上述の「開発対象のソフトウェアを生産する方法」によってソフトウェアを生産するために用いられるソフトウェアコードの雛型としてのソフトウェア、及び当該ソフトウェアが搭載された記録媒体或いは当該ソフトウェアが搭載された装置（ハードウェア）としても実現される。

【0031】

さらに、本発明は、上述の「開発対象のソフトウェアを生産する方法」による、要件から抽出した情報（ドキュメント（紙、データ））の抽出方法として、またかかる抽出方法によって抽出された情報（ドキュメント（紙、データ））として、さらには当該抽出された情報の使用方法として、或いは、これらの情報が搭載された情報記録媒体として、または情報の抽出方法／使用方法がコード化されたソフトウェア、当該ソフトウェアが搭載された記録媒体／装置（ハードウェア）として、いずれも実現することができる関連づける情報とを備えるソフトウェア開発要件から抽出した情報として実現してもよい。

【0032】

なお、論理体については、同一出願人による特願2004-272400を参照し、これを引用し、開示の一部とする。

【発明の効果】

【0033】

本発明によれば、単語単位の宣言実行モジュールからなるプログラムにおける、出力データを生成するための宣言実行モジュールの処理を、無駄な繰り返しを排除して最少実行回数で完了し、かつ、プログラムサイズを小さくすることが可能となる。より具体的には、下記の点が可能になる。

【0034】

プログラムの要件を単語単位の宣言した宣言から、実行すべき単語単位の宣言実行単位を決定し、それらの単語単位の宣言実行単位の半順序関係を定義することが可能となる。

【0035】

最適順序化によって不要となる単語単位の宣言実行単位、たとえば経路作用要素は削除する。要件の意味を維持するために実装することが必要な順序情報である経路作用要素の実行条件については、他の単語単位の宣言実行単位の半順序関係定義に反映することによって、要件の意味を保持することができる。

【0036】

こうして得られた単語単位の宣言実行単位群（経路作用要素を含めない）に対して、トポロジカル・ソートを行って、単語単位の宣言実行単位群を最適順序に並び替える。これにより、無駄な繰返しを回避し、最少実行回数でプログラムを実行することが可能となる。

【発明を実施するための最良の形態】

【0037】

本発明は、従来のLyee構造プログラムの分割単位をなくし、プログラム全体に対してトポロジカル・ソートを行うことによって、前述の従来技術の問題を改善するものである。本発明によって、Lyee方法論によって開発されるソフトウェアの実行速度の向上、プログラムサイズの改善を実現する。以下、図面を参照しながら、本願の発明の具体的な実施形態について説明する。Lyee構造プログラムに対してトポロジカル・ソートを行う技術については同一出願人による特許文献6を引用し、これを開示の一部となす。

【0038】

I章では、本発明の統合の対象であるLyeeソフトウェアの分割構造がどのようなものか、概要を説明する。II章では、基本構造を統合する具体的な方法を述べる。III章では、本発明を用いて基本構造を統合するプログラムの事例とその効果を述べる。

< I章 Lyeeソフトウェアの分割構造のしくみ >

分割構造をもった従来のLyee構造プログラムを統合するためには、まず、その分割構造のしくみについて理解する必要がある。基本構造への分割は、単語がどのようなグループ単位にどのような条件で出力されるべきなのか、という要件を捉えて宣言したものである。単語単位の宣言の要素に入出力属性という項目があったが、入出力の区別の他に、各単語がどの入力または出力グループ（論理体）に属するのか、その入出力グループがどのような条件で入出力されるのか、も宣言されるべき要件である。様々な処理を表わす要件定義から、この入出力に関する要件情報を、どのように正確に取り出すかについての詳細は、同一出願人による特願2004-272400を引用しこれを開示の一部とする。ここでは、基本構造への分割の意味について、本発明の説明に必要と思われる概要のみを述べる。

1. 基本構造への分割の意味

分割構造の意味

Lyeeソフトウェアの分割構造は、最小単位のモジュールである宣言実行モジュールが、一定の規則に従って、グループ化されている、という構造である。その構造を、図1(a)及び(b)に示す。

1) 同期範囲

プログラムを構成するモジュールは、まず、図1(a)で「同期範囲」と示した単位にグループ化されている。同期範囲とは、ユーザによるプログラムへの処理実行指示であるイベント（たとえばボタンを押す、メニューを選択する、などのコマンド）によって、コンピュータが実行するように要件で定められているひとまとまりの処理の範囲のことである。言い換えれば、同期範囲とは、同じイベントを実行条件として実行されるモジュールのグループである。L y e eでは、この同期範囲の処理が完了することを「同期」と呼ぶ。同期範囲の処理が完了するとは、イベントに対して要件で定められている出力単語の生成と出力を全て完了することである。

2) 基本構造

次に、同期範囲内のモジュールは、図1(a)の101に示すように、基本構造単位にグループ化されている。コンピュータの入出力処理は、データ単位ではなく、データの集合単位に行われるが、同時に同一の定義体に対して入力または出力処理される単語の集合をL y e eでは論理体と呼んでいる。論理体とは、画面、ファイル、帳票、電文など、単語の値を保持する単位である。同一の出力論理体に関与するモジュールをグループ化したものが基本構造である。プログラムの役割は、ユーザによるイベントが指定する出力単語の値を生成、出力することであるが、1つのイベントによる同期範囲の中で行う出力は1つとは限らない。単語の値の生成結果を画面に表示（出力）する以外に、生成結果をファイルへ保存（出力）する、などのように、多くの場合、1つの同期範囲内で2つ以上の出力論理体出力され、同期範囲は複数の基本構造にグループ化される。

3) パレット

最後に、基本構造内のモジュールは、その種類別に3つにグループ化される。基本構造内のモジュールのグループを、L y e eではパレットと呼んでいる。3つのパレットを、それぞれW02パレット、W03パレット、W04パレットと呼ぶ。W02パレットは、入力単語の入力にかかわるモジュールの集合である。W03パレットは出力単語の定義式の計算条件の判定にかかわるモジュールの集合である。W04パレットは出力単語の定義式の実行による値の生成と、値の出力にかかわるモジュールの集合である。

【0039】

次に基本構造の実行順序について説明する。

1) 同じ同期範囲内

基本構造間には端点始点関係がある。従って、深さ優先探索で次に実行すべき基本構造が決まる。同じ同期範囲内の宣言実行モジュールに対してトポロジカルソートを実行すれば実行順序が決まる。

2) 同期範囲の実行順序

どの同期範囲が実行されるかは、どのイベントがユーザによって発せられたかによる。たとえば、画面に処理Aを実行するボタンと、処理Bを実行するボタンがあり、ユーザ

がこれらのボタンのいずれか1つを押下することによって、プログラムが実行する処理内容（すなわち、同期範囲）が決定する。このように、プログラム全体は、異なるイベントを実行条件にする複数の同期範囲によって構成されているので、プログラム全体の宣言実行モジュールに対して、単純にトポロジカルソートを行うことができない。II章で、2つのイベントによる2つの同期範囲を持つ仕様を例に用いて、異なる同期範囲の宣言実行モジュールに対して統合してトポロジカルソートを行う方法を説明する。

2. トポロジカルソートの対象

トポロジカルソートによって最適順序化する、その対象は、Lyeeソフトウェアの構成要素である宣言実行モジュールが実行している処理である。宣言実行モジュールは、2つの種類があり、1つは論理要素で、単語の値を成立させるためのモジュールである。もう1つは作用要素で、入出力処理など、値の成立以外の処理を行うモジュールである。

【0040】

同一出願人による特許文献6は、値を成立させる論理要素を有向グラフおよび隣接行列の要素として取扱い、数学的モデルによって表わし、トポロジカルソートの実施形態を開示している。作用要素も、論理要素と同じく、有向グラフおよび隣接行列の要素として取扱うことができ、トポロジカルソートの対象とすることができる。論理要素と作用要素の大きな違いは、論理要素は1つの単語のみに作用（値の生成という作用）するが、作用要素は同時に2つ以上の単語に対して作用する点だけである。たとえば、経路作用要素（次に実行する基本構造を指定する）は、基本構造という単語の集合に対して作用し、出力作用要素（出力単語を出力する）は、出力論理体という単語の集合に対して作用する。

【0041】

以下に、宣言実行モジュールを順次考察しながら、トポロジカルソートの対象にする方法を述べる。

「論理要素」

『L2』

L2は、入力単語の属性が要件に合致するかのチェックを行う宣言実行モジュールである。いかなる条件下にあっても、L2の処理が実行できる条件は、入力単語の値が、媒体からメモリ領域内に取り込まれている（入力済みである）ことである。従って、トポロジカルソートの対象とするL2処理は、始点がL2の値の入力を行うI2だと考えれば良い。

【0042】

イベントのコマンド（ボタンやメニュー）も入力単語の1つとして扱う。

【0043】

『L3およびL4』

現状のLyeeソフトウェア構造では、出力単語の生成処理は、L3（計算条件式の実行）とL4（定義式の実行）の2つのモジュールによって実現している。さらに、出力単語aが複数の定義式を持つ場合（このような場合の単語を等価単語と呼ぶ）は、定義式と計算条件の組の数だけ、L3およびL4を設ける。

【0044】

出力単語aが複数の定義式を持ち（このような場合の単語を等価単語と呼ぶ）、下記のように宣言されているとする。

【0045】

＜定義式＞ ＜計算条件＞

(1) $b + c$ $e > 10$

(2) $b - d$ $e \leq 10$

等価単語は複数の定義式を持っているが、出力単語に必ず1つだけ値が与えられなければならない。従って、等価単語の定義式の計算条件は、排他的かつ完備でなければならない。

【0046】

いかなる条件下にあっても、単語aの生成処理が実行できる条件は、単語aの全ての定義式と計算条件式に用いられている全ての始点単語、が決定済みであれば良い。すなわち、トポロジカルソートの対象は、単語b、c、d、eの定義式が先に実行されていれば良い。

【0047】

従って、トポロジカルソートの対象となるのは、出力単語ごとの、L3およびL4処理の組の全てで、始点は、それらの定義式と計算条件に始点として用いられている入力単語のL2処理、および、出力単語のL4とL3処理の組である。

「作用要素」

『入力作用要素I2』

I2は、媒体からメモリ内へ値を取り込む（入力）処理を行うモジュールである。従って、いかなる条件下においても、I2の処理が実行できる条件は、無条件である（すなわち、他の処理に依存しない）。トポロジカルソートの対象となるI2処理は、始点をもたない。（ただし、ファイルからの入力の場合は、ファイルにアクセスするための

キーとなるキー単語の値が成立している必要がある)

『出力作用要素O 4』

O 4は、Lyeeソフトウェア領域からメモリ上の出力バッファへ単語の値を書き込む(出力)処理を行うモジュールである。従って、いかなる条件下においても、O 4の処理が実行できる条件は、出力対象の出力単語の、値が成立済み(すなわち、1つのL 4が実行済み)であることである。トポロジカルソートの対象となるO 4処理は、始点が出力対象の出力単語だと見なせる。

【O O 4 8】

『構造作用要素S 4』

S 4は、単語の値を記録する領域と、処理の結果(たとえば、出力が正常に完了した、出力が正常に完了しなかったなど)を記録する領域を初期化(初期値を記録する)する処理を行うモジュールである。従来の繰り返し構造をもつLyeeにおける、単語の値の領域への上書き禁止と初期化は、繰り返し構造のメカニズムのための手段であった。従って、プログラム全体にトポロジカルソートを行って、最適順序化したLyee構造においては、単語の値の領域の初期化は不要となる。

【O O 4 9】

一方、処理結果の記録領域の初期化は、繰り返し構造のメカニズムのための処理ではないので、統合Lyee構造にも必要な処理である。

【O O 5 0】

『経路作用要素』

経路作用要素は、次に実行するパレットを指定する処理を行うモジュールである。経路作用要素は、プログラムの分割単位である、パレット、およびそのパレットの集合である基本構造を、リンクする役割を果たしている。

【O O 5 1】

〈基本構造内のパレット間遷移を行う経路作用要素〉

〈基本構造間の遷移を行う経路作用要素〉

以下に例を上げて、基本構造から別の基本構造の実行を指定する経路作用要素の意味を説明する。

(例1)

図2は、あるシステムの画面で、ユーザがデータ項目cおよびデータ項目dにデータを入力して実行ボタンのaまたはbを押すことによって、データ項目gに値を得るため

のシステムである。701はデータ項目cの入力データフィールド、702はデータ項目dの入力データフィールド、703はデータ項目gの出力データフィールド、704は実行ボタンa、705は実行ボタンbである。実行ボタンaが押されるときは、gの値は定義式 $c + d$ で計算され、実行ボタンbが押されるときは、gの値は定義式 $c \times d$ によって計算される。実行ボタンのaおよびbは、両方が同時に押されることはないので、gの値の生成のための2つの定義式のどちらが実行されるかの条件となっている。

【0052】

図2の画面のプログラムを、基本構造の分割単位に表すと図3のようになる。図3のBS1、BS2、BS3はそれぞれ基本構造である。基本構造BS1は、図2に示した画面への出力論理体の基本構造である。BS1は、実行ボタンである入力単語aと入力単語b、入力データ項目である入力単語bと入力単語c、出力データ項目である出力単語g、経路作用要素 X_r を含んでいる。

【0053】

基本構造BS2は、実行ボタンaが押されたとき、出力データ項目gの値を式 $c + d$ によって計算するための基本構造（媒体がファイル）である。式 $c + d$ の結果を記録する単語eを含んでいる。基本構造BS3は実行ボタンbが押されたとき、出力データ項目gの値を式 $c \times d$ によって計算するための基本構造（媒体がファイル）である。式 $c \times d$ の計算結果を記録する単語fを含んでいる。

【0054】

画面にユーザがデータを入力した（BS1が実行された）後に、出力単語gの値を計算するための基本構造はBS2またはBS3であるが、どちらが実行されるべきかを指定するのが経路作用要素 X_r である。2つの基本構造に所属する単語は、それぞれボタンaとbのどちらが押されたかによって実行が決まるので、経路作用要素 X_r は、実行ボタンaが押された場合（単語a＝真、値がある）はBS2を、実行ボタンbが押された場合（単語b＝真、値がある）はBS3を、次に実行する基本構造として指定する。また、実行が次の基本構造に移るのは入力データがそろったときであるので、単語cおよび単語dに値があることも、経路作用要素 X_r の実行条件となる。

【0055】

基本構造BS2またはBS3の実行によって、出力単語gの値が単語eまたは単語fに成立する。出力データを画面に出力するための基本構造BS1の出力単語gは、単語eに値が成立したとき（e＝真）は $g = e$ 、単語fに値が成立したとき（f＝真）は $g = f$ 、と定義される。

【0056】

上記のような単語の定義を整理すると下記の表1のとおりである

【0057】

【表1】

単語	入出力属性	定義式	定義式実行条件
a	入力	—	—
b	入力	—	—
c	入力	—	—
d	入力	—	—
x_r	(経路作用要素)	BS 2を指定	(1) $a = \text{真}$ (ボタンaが押され、aに値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
		BS 3を指定	(2) $b = \text{真}$ (ボタンbが押され、bに値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
e	出力	$c + d$	x_r がBS 2を指定
f	出力	$d \times d$	x_r がBS 3を指定
g	出力	e	$e = \text{真}$ (eに値がある)
		f	$f = \text{真}$ (fに値がある)

『制御モジュール』

従来のLyee構造は、分割構造があり、分割単位に繰り返し処理が行われていたために、パレット内の各モジュールをコールして実行させるパレット関数（パレットの数だけ設置）と、それらのパレット関数自体をコールして実行させるパレット連鎖関数（プログラムに1つ）、の2種類の制御モジュールが必要であった。

【0058】

トポロジカルソート後は、プログラム全体が統合され、順序化されるので、プログラム全体に1つ、プログラム内のモジュールをコールする制御モジュールがあれば良い。

II章 基本構造の統合とトポロジカルソート

例1の要件を用いて、トポロジカルソートの具体的手順を説明する。

1. 基本構造の隣接行列

図4は、図3の単語の関係を有向グラフによって示した図である。基本構造BS 1に属する入力単語である単語a、単語b、単語c、単語dは、有向グラフによって表される単語のネットワークの最端始点となる。同じく基本構造BS 1に属する経路作用要素である単語 x_r は、上記表1に示したとおり、値が成立（次に実行する基本構造決定の真偽判定が成立）するためには、単語aまたは単語bと単語cと単語dが必要であるので、これらの4つの単語を始点とする矢印によって4つの単語とリンクされる。基本構造BS 2に属す単語e、および、基本構造BS 3に属す単語fは、値が成立するために、単語

c、単語dおよび単語 x_r が必要であるので、これらの3つの単語を始点とする矢印によって各々3つの単語とリンクされる。基本構造BS1に属す出力単語gは、値が成立するために、単語eまたは中間単語fが必要であるので、これらの2つの単語を始点とする矢印によって2つの単語とリンクされる。

【0059】

次に、図4で有向グラフで示した例1のシステムを隣接行列を用いて表す。最初に図4の、入力単語を扱った基本構造BS1 (901)を表した隣接行列F1を下記の数1に示す。

基本構造BS1 (入力単語) の隣接行列

【0060】

【数1】

$$F1 = \begin{matrix} & \begin{matrix} a & b & c & d & x_r \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ x_r \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

BS1 (入力単語) の要素は単語a、単語b、単語c、単語d、単語 x_r であるから、F1の行列はこれらの要素によって構成される。入力単語である単語a、単語b、単語c、単語dは、始点単語を持たないので、終点単語a、b、c、dのそれぞれの横列 (始点単語a、b、c、d、 x_r との交点) は総て始点単語として用いないことを示す「0」となる。経路作用要素である単語 x_r は、図4の有向グラフに示したとおり、始点として単語a、単語b、単語c、単語dを持つので、終点単語 x_r の横列は、これらの始点単語との交点は始点であることを示す「1」となる。

【0061】

次に、基本構造BS2およびBS3 (以下、基本構造BS2 & BS3と記す) を合わせたものを表す隣接行列F2を下記の数2に示す。

基本構造BS2 & BS3の隣接行列

【0062】

【数2】

$$F2 = \begin{matrix} & e & f \\ \begin{matrix} e \\ f \end{matrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{matrix}$$

BS2 & BS3の要素は単語eと単語fのみであるので、F2の行列はこれら2つの要素で構成される。単語eと単語fはお互いに独立している（すなわち、どちらの単語も始点単語としても一方の単語を用いていない）ので、いずれの交点も「0」となる。

【0063】

基本構造BS1の出力単語を扱った集合の隣接行列F3を、下記の数3に示す。

基本構造BS1（出力単語）の隣接行列

【0064】

【数3】

$$F3 = \begin{matrix} & g \\ g & [0] \end{matrix}$$

BS3の要素は単語gのみであるので、F3の行列は単語g1つのみで構成される。単語gは始点として自身を用いないので、交点は「0」となる。

【0065】

以上で、例1のシステムを構成する基本構造単位に、同一基本構造内の要素間の関係を隣接行列に表した。

2. 結合隣接行列による基本構造の統合

次に、これらをシステム全体として1つの隣接行列として表すために、各基本構造の要素である単語と他の基本構造の要素である単語との関係を示すことによって基本構造をリンクする結合隣接行列（connection matrix）について説明する。

【0066】

任意の基本構造の単語と他の基本構造の単語の関係は、図4に示したように終点単語と始点単語の関係によって表わされている。基本構造同士は、それに属する単語の関係によってリンクされていると言える。図4から以下のことが分かる。

- 1) 基本構造BS1（入力）の単語を終点単語としたとき、いずれの他の基本構造の単語も、BS1（入力）の単語より後に現れるので、始点単語となりえない。
- 2) 基本構造BS2 & 3の単語を終点単語としたとき、BS2 & 3の単語より前に現れ

るBS1（入力）の単語は始点単語となりえるが、BS1（出力）の単語は始点単語となりえない。

- 3) 基本構造BS1（出力）の単語を終点単語としたとき、BS2&3およびBS1（入力）のいずれの単語も、BS1（出力）の単語より前に現れるので、始点単語となりえる。

【0067】

上記の考察の結果、基本構造をリンクするための結合隣接行列として重要なのは、任意の基本構造と、それに属す単語の始点単語になりえる単語を含む基本構造との関係を表す結合隣接行列であることがわかる。なぜなら、有向グラフで示されるように、単語間の関係は始点単語から終点単語へと向かう関係のみが存在するがゆえに、基本構造間の関係も、それと同様に始点単語を含む基本構造から終点単語を含む基本構造へと向かう関係のみが存在し、逆の関係は存在しない（そのような基本構造間の関係を表す結合隣接行列は、すべての交点が0となる）からである。

【0068】

以下に、例1の基本構造をリンクする（始点単語を含む基本構造から終点単語を含む基本構造へと向かう関係を表す）結合隣接行列を1つずつ考察する。

【0069】

数4は、基本構造BS1（入力）（始点単語）から基本構造BS2&BS3（終点単語）への関係を示す結合隣接行列FC1である。

【0070】

【数4】

$$FC1 = \begin{matrix} & \begin{matrix} a & b & c & d & x_r \end{matrix} \\ \begin{matrix} e \\ f \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

隣接行列FC1は、基本構造BS2&BS3の要素である単語eと単語fを終点単語においたとき、BS1（入力）の要素である単語a、単語b、単語c、単語d、単語x_rが始点単語として用いられる状態を表し、2つの基本構造の関係を示している。図4で示したように、単語eおよび単語fは、単語c、単語d、単語x_rを始点単語に用いる。従って、FC1において、終点単語eおよび終点単語fの行と、始点単語c、d、x_rとの交点は「1」となり、その他の始点単語との交点は「0」となる。

【0071】

数5は、基本構造BS1（入力）および基本構造BS2 & BS3（始点単語）から、基本構造BS1（出力）（終点単語）への関係を示す隣接行列FC2を表している。

【0072】

【数5】

$$FC2 = g \begin{matrix} & a & b & c & d & e & f \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix}$$

隣接行列FC2は、基本構造BS1（出力）の要素である単語gを終点単語においたとき、有向グラフにそれ以前に現われるBS1（入力）、BS2、BS3の要素である単語a、単語b、単語c、単語d、単語x、単語e、単語f、が始点単語として用いられる状態を表して、2つのグループの関係を表している。図4で示したように、単語gは、単語eと単語fを始点単語に用いる。従って、FC2において、終点単語gの行と、始点単語eおよびfとの交点は「1」となり、その他の始点単語との交点は「0」となる。

【0073】

以上が、例1のシステムを構成する基本構造間のリンクに意味を持つ結合隣接行列であった。システム全体として1つの隣接行列へ統合する過程の説明を簡便にするために、下記に、関係が存在しない基本構造間の結合隣接行列も示す。数6は、基本構造BS2 & BS3（始点単語）から基本構造BS1（入力）（終点単語）の関係がないことを示す結合隣接行列FC3である。

【0074】

【数6】

$$FCJ = \begin{matrix} & e & f \\ \begin{matrix} a \\ b \\ c \\ d \\ x_1 \end{matrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{matrix}$$

数7は、基本構造BS1（出力）（始点単語）から基本構造BS1（入力）（終点単語）の関係がないことを示す結合隣接行列FC4である。

【0075】

【数7】

$$FC4 = \begin{matrix} & g \\ a & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ b & \\ c & \\ d & \\ x_r & \end{matrix}$$

数8は、基本構造BS1（出力）（始点単語）から基本構造BS2 & BS3（終点単語）の関係がないことを示す結合隣接行列FC5である。

【0076】
【数8】

$$FC5 = \begin{matrix} & g \\ e & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ f & \end{matrix}$$

上記に示した隣接行列F1からF3、結合隣接行列FC1からFC5を用いて、例1のシステムを1つの隣接行列Fとして表したものが数9である。

【0077】
【数9】

$$F = \begin{bmatrix} F1 & 0 & 0 \\ FC1 & F2 & 0 \\ FC2 & & F3 \end{bmatrix} = \begin{matrix} & a & b & c & d & x_r & e & f & g \\ \begin{matrix} a \\ b \\ c \\ d \\ x_r \\ e \\ f \\ g \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

上記数式で、「F=」の右辺に示された、F 1、F 2、F 3、F C 1、F C 2と0（行列の交点が総て0である隣接行列で、F C 3、F C 4、F C 5に相当する）で構成された行列は、基本構造単位の隣接行列と結合隣接行列がどのように統合されてシステム全体を表わすかを示している。次に、それらの要素を単語に変えたものが、その右辺にあるa、b、c、d、x_r、e、f、gを行列の要素とする隣接行列である。F 1、F 2、F 3、F C 1、F C 2、F C 3、F C 4、F C 5がどのように統合されてFを構成するのか、さらにわかりやすく示したのが図5である。図5の1 0 0 1の部分でF 1、1 0 0 2がF 2、1 0 0 3がF 3、1 0 0 4がF C 1、1 0 0 5がF C 2である。行列の交点が総て0である結合隣接行列は、F C 3が1 0 0 6、F C 4が1 0 0 7、F C 5が1 0 0 8を構成する。

【0078】

以上のようにして、2つ以上の基本構造から成る1つのシステムは、1つの隣接行列で表すことができる。

【0079】

以上でシステム全体を1つの隣接行列で表わすことができたので、このシステムの隣接行列に対して、同一出願人による特許文献6で開示されるトポロジカルソートを行って、総ての単語の値の状態が最少実行回数で確定（すなわち総ての出力単語の生成が完了）するような最適順序に単語単位のプログラムの並び替えを行う。

【0080】

例1のシステムの隣接行列Fをみれば、図6に示すように、値が1である始点単語と

の交点は総て左下三角形 (1 3 0 1) の内にある。I 章で述べたように、これは単語が最適順序に並んでいることを意味する。従って、数 9 の隣接行列 F は、すでにトポロジカルソート済みであって、初期値の単語の状態ベクトルを与えれば、最少実行回数で出力単語の生成を完了することができる単語の順序になっているといえる。

【0081】

ここで、トポロジカルソートを施す効果について補足する。トポロジカルソートによって最適実行順序に並んだ単語単位のプログラムは、1 巡処理で総ての出力単語の値を生成することができる。しかし、システム全体に対して言及する場合には、より正確には「最少実行回数」で実現できるということが適切である。なぜなら、システム全体でみたときには、集計処理のように、一時的記録領域を置くことによって同じ計算式を繰り返し利用する必要がある処理を含む場合があるからである。もちろん、この場合の「繰り返し」は、無駄な繰り返しではない。集計するデータの数マイナス 1 が最少繰り返し数になる。このような集計処理も、結合隣接行列を用いて 1 つの隣接行列に表わすことができる。

【0082】

なお、ここでは探索技法の一例としてトポロジカル・ソートを例にとり説明したが、他の探索の技法を用いても良い。また、隣接行列定義については、本稿では理解の容易さを助けるために入れたが、本質的には必須のステップではない。つまり、隣接行列を定義することなく、宣言間関係の規定から、直接に、或いは有向グラフ作成を介して、トポロジカル・ソート等の探索に至ることも本発明の思想の範囲内であり、本発明の目的とするところを実現し得る。

3. プログラムの隣接行列の検証

本セクションでは、前セクションで示したシステムの隣接行列 F が、出力単語の値を生成する関数として機能し、しかも、トポロジカル・ソート済みであるので、最少実行回数で総ての出力単語の生成を完了できることを、同一出願人による特許文献 6 で開示される単語の状態ベクトルをかける計算操作によって検証する。出力単語 g の値が決定されれば、隣接行列 F は関数として機能すると言える。また、例 5 のシステムは、前述の集計処理のような繰り返し処理を含まないので、1 巡処理で出力単語 g の値の状態が確定すれば、最少実行回数で処理が完了したと言える。

【0083】

例 1 のシステムにおいて、ユーザが入力を行う前の単語の値の状態、すなわち単語の状態ベクトル X の初期値は、同一出願人による特許文献 6 で開示されるように総ての要素が「null (未決定)」で、下記の数 10 のようになる。

【0084】

【数 10】

$$X = \begin{bmatrix} a \\ b \\ c \\ d \\ x_r \\ e \\ f \\ g \end{bmatrix} \begin{bmatrix} \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \end{bmatrix}$$

数 9 の隣接行列 F (すなわち例 1 のシステム) に、数 10 の状態ベクトル X をかける計算処理を施す (すなわち、例 1 のシステムに入力を行う)。F X の計算処理によって変化する単語の値の状態を 1 つの終点単語ごとに追って説明する。

(1) 1 巡目処理中の終点単語 a の計算

1 巡目処理中の終点単語 a の計算は、次のようになる。

$$\begin{aligned} a &= 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\ &= (+1) \end{aligned}$$

入力単語 a は始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、null から (+1) へ変化する。すなわち、これは入力によって値が確定したことを表わしている。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 2 のとおりである。

【0085】

【表 2】

	a	b	c	d	x _r	e	f	g
値の状態	+1	null	null	null	null	null	null	null

(2) 1 巡目処理中の終点単語 b の計算

1 巡目処理中の終点単語 b の計算は、表 2 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 b &= 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 b も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1 巡処理中の単語 b 行完了後の各単語の値の状態は、以下の表 3 のとおりである。

【0086】

【表 3】

	a	b	c	d	x _r	e	f	g
値の状態	+1	+1	null	null	null	null	null	null

例 1 のシステムでは、単語 a と単語 b は画面上の 2 者択一の指示ボタンであるので、どちらか一方が選択されることによって、双方の値の状態が確定することになる。

(3) 1 巡目処理中の終点単語 c の計算

1 巡目処理中の終点単語 c の計算は、表 3 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 c &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 c も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1 巡処理中の単語 c 行完了後の各単語の値の状態は、以下の表 4 のとおりである。

【0087】

【表 4】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	null	null	null	null	null

(4) 1巡目処理中の終点単語 d の計算

1巡目処理中の終点単語 d の計算は、表 4 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 d &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 d も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 d 行完了後の各単語の値の状態は、以下の表 5 のとおりである。

【0088】

【表 5】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	null	null	null	null

(5) 1巡目処理中の終点単語 x_r の計算

1巡目処理中の終点単語 x_r の計算は、表 5 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 x_r &= 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 x_r の始点単語となる単語 a、b、c、d の値の状態がすでに確定した、すなわち、表 1 に示した経路作用要素である単語 x_r の定義式実行条件 (1) および (2) が満たされたので、単語 x_r の値の状態もnullから確定済みの(+1)へ変化する。その結果、1巡処理中の単語 x_r 行完了後の各単語の値の状態は、以下の表 6 のとおりである。

【0089】

【表 6】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	null	null	null

(6) 1 巡目処理中の終点単語 e の計算

1 巡目処理中の終点単語 e の計算は、表 6 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 e &= 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 e の始点単語となる単語 c、d および x_r の値の状態がすでに確定済みであるので、すなわち、定義式 $c + d$ の始点単語の値が決定済み、かつ経路作用要素 x_r が次に実行する基本構造を指定したので、単語 e の値の状態も null から確定済み (+1) へ変化する。その結果、1 巡処理中の単語 e 行完了後の各単語の値の状態は、以下の表 7 のとおりである。

【0 0 9 0】

【表 7】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	null	null

(7) 1 巡目処理中の終点単語 f の計算

1 巡目処理中の終点単語 f の計算は、表 7 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 f &= 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 f の始点単語となる単語 c、d および x_r の値の状態がすでに確定済みであるので、単語 f の値の状態も null から (+1) へ変化する。その結果、1 巡処理中の単語 f 行完了後の各単語の値の状態は、以下の表 8 のとおりである。

【0 0 9 1】

【表 8】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1	null

例 1 のシステムでは、単語 e と単語 f はそれぞれの実行条件が 2 者択一の条件であるので、どちらか一方の条件が真と確定すれば、他方の条件は偽と確定し、双方の値の状態が確定することになる。たとえば、BS 2 が指定されれば、単語 e の定義式実行条件 = 真が確定して単語 e に値が与えられることによって値の状態が確定する。一方、単語

f は、定義式実行条件＝偽が確定して単語 f に値が与えられないことによって値の状態が確定する。従って、経路作用要素がどちらか一方の基本構造を指定したとき、単語 e も単語 f も値の状態が確定済みとなるのである。

(8) 1 巡目処理中の終点単語 g の計算

1 巡目処理中の終点単語 g の計算は、表 8 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 g &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 g の始点単語となる単語 e および f の値の状態がすでに確定済みであるので、単語 g の値の状態も null から (+1) へ変化する。その結果、1 巡処理中の単語 g 行完了後の各単語の値の状態は、以下の表 9 のとおりである。

【0 0 9 2】

【表 9】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1	+1

以上で隣接行列 F の総ての単語行の計算処理が 1 巡したことになる。従って、表 9 が F X の結果となる。下記の数 1 1 の状態ベクトル X_1 が F X の結果である。

【0 0 9 3】

【数 1 1】

$$X_1 = \begin{bmatrix} a \\ b \\ c \\ d \\ x_r \\ e \\ f \\ g \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

以上のように、隣接行列Fは、トポロジカルソートによって単語単位プログラムが最適順序で実行されるように並んでいるので、1巡の計算処理の実行で総ての単語の値の状態が確定し、出力単語gの値を成立することができた。

4. 経路作用要素の排除

経路作用要素の実行条件は、経路作用要素によって指定される基本構造に属する全単語の集合（1つ以上からなる）の定義式を実行するための条件と等しい。なぜなら、その基本構造の単語の総ては、その経路作用要素の条件が満されたときに実行されるからである。従って、経路作用要素の実行条件は、経路作用要素によって指定される基本構造に属す単語の条件に移すことができる。

【0094】

例1のシステムの場合で具体的に述べると次のようになる。まず、経路作用要素である単語 x_r と、単語eおよび単語fの定義式と定義式実行条件は下記の表10の通りであった。

【0095】

【表10】

単語	定義式	定義式実行条件
x_r	BS 2を指定	(1) $a = \text{真}$ (ボタン a が押され、 a に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
	BS 3を指定	(2) $b = \text{真}$ (ボタン b が押され、 b に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
e	$c + d$	x_r が BS 2 を指定
f	$d \times d$	x_r が BS 3 を指定

従って、 x_r の条件 (1) 「 $a = \text{真}$ かつ、 $c = \text{真}$ および $d = \text{真}$ 」を、この条件が成立するとき指定する基本構造 BS 2 に属する単語 e の実行条件「 x_r が BS 2 を指定」と置き換え、条件 (2) 「 $b = \text{真}$ かつ、 $c = \text{真}$ および $d = \text{真}$ 」をこの条件が成立するとき指定する基本構造 BS 3 に属する単語 f の実行条件「 x_r が BS 3 を指定」と置き換える、ということになる。この結果、単語 e および単語 f の定義式、定義式実行条件は下記の表 1 1 のようになる。

【0096】

【表 1 1】

単語	定義式	定義式実行条件
e	$c + d$	$a = \text{真}$ (ボタン a が押され、 a に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
f	$c \times d$	$b = \text{真}$ (ボタン b が押され、 b に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$

表 1 1 のように、経路作用要素の実行条件を、指定先の基本構造の単語の実行条件に置き換えると、BS 1 と BS 2 および BS 3 のリンクは、BS 2 および BS 3 の単語の実行条件によって成立するので、経路作用要素は単に、次の基本構造の実行に移る条件が整ったこと（入力単語 c および d に値があり、かつボタン a または b が押された）を示すだけの意味しか持たなくなる。これは、経路作用要素を取り除いてもシステムの意味に変化が生じないことを意味する。従って、経路作用要素の実行条件を、その条件の成立によって指定される基本構造の単語の実行条件に置き換える操作を行った後、システム全体の隣接行列から、経路作用要素を除くことができる。

【0097】

図 7 は、経路作用要素を取り除いて、例 1 のシステムの単語の関係を有向グラフで表したものである。図 7 では、経路作用要素である単語 x_r がなくなり、単語 e および単語 f と始点単語との関係は、上記の表 1 1 で定義された定義式と定義式実行条件に従って示されている。すなわち、単語 e は、始点単語として単語 a （定義式実行条件「実行ボタン a が押された」）、単語 c および単語 d （定義式「 $c + d$ 」および、定義式実行条件

「 $c = \text{真}$ および $d = \text{真}$ 」)を持つ。単語 f は、始点単語として単語 b (定義式実行条件「実行ボタン b が押された」)、単語 c および単語 d (定義式「 $c + d$ 」および、定義式実行条件「 $c = \text{真}$ および $d = \text{真}$ 」)を持つ。単語 e および単語 f から単語 g への関係は図 4 と変らない。

【0098】

経路作用要素を取り除いた例 1 のシステムを、図 7 の有向グラフに従って隣接行列に表わせば、以下のようになる。数 1 2 の $F1'$ は、基本構造 $BS1$ (入力) の隣接行列である。経路作用要素である単語 x_r が要素から削除されている。

【0099】

【数 1 2】

$$F1' = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

数 1 3 の $F2'$ は、基本構造 $BS2$ & $BS3$ の隣接行列である。

【0100】

【数 1 3】

$$F2' = \begin{matrix} e & f \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{matrix}$$

数 1 4 の $F3'$ は、基本構造 $BS1$ (出力) の隣接行列である。

【0 1 0 1】

【数 1 4】

$$F3' = \begin{matrix} g \\ [0] \end{matrix}$$

数 1 5 の $FC1'$ は、基本構造 $BS1$ (入力) から基本構造 $BS2$ & $BS3$ へ の関係を示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語 X_r が削除されている。単語 e および単語 f は、表 1 1 に示したように経路作用要素の実行条件を移したことで、それぞれ新たに単語 a 、単語 b を始点単語として持つことになった。従って、終点単語 e は、始点単語 c および d の他に単語 a との交点が 1 となっている。終点単語 f は、始点単語 c および d の他に単語 b との交点が 1 となっている。

【0 1 0 2】

【数 1 5】

$$FC1' = \begin{matrix} & a & b & c & d \\ \begin{matrix} e \\ f \end{matrix} & \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

数16のFC2'は、基本構造BS2 & BS3から基本構造BS1（出力）へ関係を示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語X_rが削除されている。

【0103】

【数16】

$$FC2' = \begin{matrix} & a & b & c & d & e & f \\ g & 0 & 0 & 0 & 0 & 1 & 1 \end{matrix}$$

数17のFC3'は、基本構造BS2 & BS3から基本構造BS1（入力）へ関係がないことを示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語X_rが削除されている。

【0104】

【数17】

$$FC3' = \begin{matrix} & & e & f \\ a & \vdots & 0 & 0 \\ b & \vdots & 0 & 0 \\ c & \vdots & 0 & 0 \\ d & 0 & 0 & \end{matrix}$$

数18のFC4'は、基本構造BS1（出力）から基本構造BS1（入力）へ関係がないことを示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語X_rが削除されている。

【0105】

【数18】

$$FC4' = \begin{matrix} & g \\ a & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ b & \\ c & \\ d & \end{matrix}$$

数19のFC5'は、基本構造BS1（出力）から基本構造BS2 & BS3 への関係がないことを示す結合隣接行列である。

【0106】

【数19】

$$FC5' = \begin{matrix} & g \\ e & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ f & \end{matrix}$$

システムの隣接行列Fは、経路作用要素の条件の置き換えと経路作用要素の削除を行うと、下記の数20に示す隣接行列F'で表わすことができる。

【0107】

【数 20】

$$F' = \begin{bmatrix} F1' & 0 & 0 \\ FC1' & F2' & 0 \\ FC2' & & F3' \end{bmatrix} = \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

上記数式で、「F'＝」の右辺に示された、F1'、F2'、F3'、FC1'、FC2'と0（行列の交点が総て0である隣接行列で、FC3'、FC4'、FC5'に相当する）で構成された行列は、基本構造単位の隣接行列と結合隣接行列がどのように統合されてシステム全体を表わすかを示している。次に、それらの要素を単語に変えたものが、その右辺にあるa、b、c、d、e、f、gを行列の要素とする隣接行列である。F1'、F2'、F3'、FC1'、FC2'、FC3'、FC4'、FC5'がどのように統合されてF'を構成するのか、さらにわかりやすく示したのが図8である。図8の1201の部分にF1'、1202がF2'、1203がF3'、1204がFC1'、1205がFC2'である。行列の交点が総て0である結合隣接行列は、FC3'が1206、FC4'が1207、FC5'が1208を構成する。

【0108】

以上のようにして、2つ以上の基本構造から成る1つのシステムを、経路作用要素を排除して、1つの隣接行列で表わすことができる。

【0109】

次に、このシステムの隣接行列に対して、トポロジカルソートを行って、総ての単語の値の状態が最少実行回数で確定（すなわち総ての出力単語の生成が完了）するような最適順序に単語単位のプログラムの並び替えを行う。隣接行列F'をみると、図9に示すように、値が1である始点単語との交点は総て左下三角形（1601）内にある。同一出願人による特許文献5で示されるように、これは単語が最適順序に並んでいることを意味する。従って、数20の隣接行列F'は、すでにトポロジカルソート済みであって、初期値の単語の状態ベクトルを与えれば、最少実行回数で出力単語の生成を完了することができる単語の順序になっているといえる。

5. 経路作用要素を排除した構造の検証

このセクションでは、前セクションで示した、経路作用要素を排除したシステムの構造を示す隣接行列 F' が、経路作用要素を排除する前の構造である隣接行列 F と同じ関数機能を持つことを検証する。以下に、隣接行列 F' に単語の状態ベクトルを与えて出力単語 g の値を求める過程を示す。 m 回目の計算処理の結果、単語の状態ベクトル X_m の値が総て確定済みとなれば、 F' も F と同等に機能する関数だと言える。また、トポロジカルソートを施したことによって、最少実行回数で出力単語の生成を完了できることも検証する。例 1 のシステムである F' も、1 巡処理で単語 g の値の状態が確定すれば、最少実行回数で実行できたと言える。

【0 1 1 0】

前述したように、単語の状態ベクトルの初期値は総ての要素が「null（未決定）」であるので、 F' に最初に与えるべき単語の状態ベクトル X' は下記の数 2 1 のようになる。

【0 1 1 1】

【数 2 1】

$$X' = \begin{bmatrix} a & \text{null} \\ b & \text{null} \\ c & \text{null} \\ d & \text{null} \\ e & \text{null} \\ f & \text{null} \\ g & \text{null} \end{bmatrix}$$

数 2 0 の隣接行列 F' （すなわち例 1 のシステム）に、数 2 0 の状態ベクトル X' をかける計算処理を施す（すなわち、例 1 のシステムに入力を行う）。 $F'X'$ の計算処理によって変化する単語の値の状態を 1 つの終点単語ごとに追って説明する。

(1) F' 1 巡目処理中の終点単語 a の計算

終点単語 a は、計算処理に X' の単語の値の状態を用いるので、1 巡目処理中の終点単語 a の計算は、次のようになる。

$$\begin{aligned}
 a &= 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 a は始点単語を持たないので、値の状態が確定し、null から (+1) へ変化する。すなわち、これは入力によって値が確定したことを表わしている。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 1 2 のとおりである。

【0 1 1 2】

【表 1 2】

	a	b	c	d	e	f	g
値の状態	+1	null	null	null	null	null	null

(2) F' 1 巡目処理中の終点単語 b の計算

終点単語 b は、計算処理に表 1 2 の単語の値の状態を用いるので、1 巡目処理中の終点単語 b の計算は、次のようになる。

$$\begin{aligned}
 b &= 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 b も始点単語を持たないので、値の状態が確定し、null から (+1) へ変化する。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 1 3 のとおりである。

【0 1 1 3】

【表 1 3】

	a	b	c	d	e	f	g
値の状態	+1	+1	null	null	null	null	null

(3) F' 1 巡目処理中の終点単語 c の計算

終点単語 c は、計算処理に表 1 3 の単語の値の状態を用いるので、1 巡目処理中の終点単語 c の計算は、次のようになる。

$$\begin{aligned}
 c &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 c も始点単語を持たないので、値の状態が確定し、null から (+1) へ変化する。その結果、1 巡処理中の単語 c 行完了後の各単語の値の状態は、以下の表 1 4 のとおりである。

【0 1 1 4】

【表 1 4】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	null	null	null	null

(4) F' 1 巡目処理中の終点単語 d の計算

終点単語 d は、計算処理に表 1 4 の単語の値の状態を用いるので、1 巡目処理中の終点単語 d の計算は、次のようになる。

$$\begin{aligned}
 d &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 d も始点単語を持たないので、値の状態が確定し、null から (+1) へ変化する。その結果、1 巡処理中の単語 d 行完了後の各単語の値の状態は、以下の表 1 5 のとおりである。

【0 1 1 5】

【表 1 5】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	null	null	null

(5) F' 1 巡目処理中の終点単語 e の計算

終点単語 e は、計算処理に表 1 5 の単語の値の状態を用いるので、1 巡目処理中の終点単語 e の計算は、次のようになる。

$$\begin{aligned}
 e &= 1 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 e の始点単語である単語 a、c、d の値の状態が確定しているので、単語 e の値の状態も null から (+1) へ変化して確定する。その結果、1 巡処理中の単語 e 行完了後の各単語の値の状態は、以下の表 1 6 のとおりである。

【0 1 1 6】

【表 1 6】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	null	null

(6) F' 1 巡目処理中の終点単語 f の計算

終点単語 f は、計算処理に表 1 6 の単語の値の状態を用いるので、1 巡目処理中の終点単語 f の計算は、次のようになる。

$$\begin{aligned}
 f &= 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 f の始点単語である単語 b、c、d の値の状態が確定しているので、単語 f の値の状態も null から (+1) へ変化して確定する。その結果、1 巡処理中の単語 f 行完了後の各単語の値の状態は、以下の表 1 7 のとおりである。

【0 1 1 7】

【表 1 7】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	null

(7) F' 1 巡目処理中の終点単語 g の計算

終点単語 g は、計算処理に表 1 7 の単語の値の状態を用いるので、1 巡目処理中の終

点単語 g の計算は、次のようになる。

$$\begin{aligned}
 g &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 g の始点単語である単語 e 、 f の値の状態が確定しているので、単語 g の値の状態も null から $(+1)$ へ変化して確定する。その結果、1 巡処理中の単語 g 行完了後の各単語の値の状態は、以下の表 18 のとおりである。

【0 1 1 8】

【表 18】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1

以上で隣接行列 F' の総ての単語行の計算処理が 1 巡したことになる。従って、表 18 の単語の値の状態が $F'X'$ の結果となる。下記の数 22 の状態ベクトル X'_1 が $F'X'$ の結果である。

【0 1 1 9】

【数 22】

$$X'_1 = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

以上のように、隣接行列 F' は、トポロジカルソートによって単語単位プログラムが最適順序で実行されるように並んでいるので、1巡の計算処理の実行で総ての単語値が確定し、出力単語 g の値を成立することができた。

【0120】

また、経路作用要素の単語を除いた隣接行列 F' も、状態ベクトルの初期値を与える（入力を行う）ことで、隣接行列 F と同様に総ての単語の値の状態が確定し、出力単語 g の値が生成できたことがわかる。従って、経路作用要素のあるシステムである隣接行列 F と、経路作用要素のないシステムである隣接行列 F' は、システムとして同等の処理機能を持つことが検証できた。

【0121】

上記で述べたように、全体システムが複数の基本構造を有する場合に、これを統合し、さらに当該統合されたものに対して（例えばトポロジカルソートのような）探索を施すことで、L y e e（登録商標）等のソフトウェアの自律的開発技術の適用に際して各段のパフォーマンスを提供する。これが例えば、統合されずに複数の基本構造をそのまま有する場合には、本願に係る発明によるほどのパフォーマンスは得られず、また、トポロジカルソート等の探索技法によって整序しなかった場合には本願に係る発明によるほどのパフォーマンスは得られない。

6. プログラムの再利用

プログラムの再利用が必要な処理は、トポロジカルソートを行っても繰り返しが行われる。

どんなプログラム（アルゴリズム）でも再帰的な方法だけによって記述することができる。手続き型のプログラミング言語（例えば、C、java など）によって書かれたどんなプログラムでも、再帰プログラムと同じ役割を果たすwhileプログラムによって書くことができる。このことは、whileプログラムは理論的には、なんでもできることを示している。

しかしながら、単語を基礎とするプログラム法では、自分自身を始点単語として使用することができない。例えば、 $a = a + b$ を単語を基礎とするプログラムでは使うことはできない。なぜなら、 a は、一度決定(decided)になると a の値は変化しないからである。したがって、再帰的使用のためには、一時的に a の値を保持する領域とクリアする方法が必要となる。保持された値を始点として使用する。この目的のために一時的に内容が保持される領域を持つことが必要である。この領域を一時的な単語と呼ぶ。クリアするために構造作用要素(Vector of Clear)が定義される。

Lyeeでは一般に、作用要素は述語構造を持つ一種の単語である。プログラムコードの構造は単語と同じである。単語との主な相違は同時にふたつあるいはもっと多くの単語を扱っていることである。その意味は外部とのインターフェースおよび制御構造のため

の経路や領域のクリアである。

【0 1 2 2】

プログラムの隣接行列（プログラム全体の行列）は分割された隣接行列とただ状態を移すだけの接続行列とを使って、1つの行列に統合化される。接続行列は、他のグループ上の単語を始点単語として使うことを示している。したがって、単語を基礎とするプログラムの数学的なモデルを使うことが可能である。それによって、手続き型の言語で表現することができるすべての通常のアigorリズムは、単語を基礎とするプログラムの構造で表現することができる。

【0 1 2 3】

次に、前述の $a=a+b$ のような、一時的な単語を必要とする集計処理の例である(例2)を用いて、接続行列による隣接行列の統合を説明する。

(例2) ファイルのある単語のそれぞれのレコード x_2 、 x_6 の値は先頭から順番に合計し、集計結果を単語 x_3 、 x_7 に集計する。

【0 1 2 4】

Lyeeでは、一度決定した値は上書きできないので、 $x_3 = x_1 + x_2$ のように記述しこれを再利用する。 x_1 は前回の x_3 の値を代入するための単語である。 x_3 の値は、次の再利用時に x_1 に代入するために、一時的な領域 x_4 に転送される。「一時的な単語」と呼ぶ。この例2を有向グラフに表すと図10に示すようになる。単語 x_1 、 x_2 、 x_3 は、 x_3 が x_4 に転送されたときにクリアされる。単語 x_5 、単語 x_6 、単語 x_7 は、それぞれ単語 x_1 、単語 x_2 、単語 x_7 の再使用である。

【0 1 2 5】

図10の再使用されるプログラムの部分の隣接行列(one path matrix)は以下に示すFIである。

【0 1 2 6】

【数23】

$$\begin{array}{c}
 x_{1,5} \quad x_{2,6} \quad x_{3,7} \\
 F1 = \begin{array}{c} x_{1,5} \\ x_{2,6} \\ x_{3,7} \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} : \text{one path matrix}
 \end{array}$$

$F1$ を再利用するため、 $F1$ と再利用された $F1$ とを統合する接続行列(connection matrix)が $F2$ と $F3$ である。これらは状態を伝達するだけの機能を有するものである。それぞれの終点単語が、始点単語としてどの単語を使用するかを示す。

【0 1 2 7】

【数 2 4】

$$\begin{array}{c}
 x_1 \quad x_2 \quad x_3 \\
 F2 = x_4 \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} : \text{connection matrix from } x_1, x_2, x_3 \text{ to } x_4 \\
 \\
 x_4 \\
 F3 = \begin{array}{c} x_5 \\ x_6 \\ x_7 \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} : \text{connection matrix from } x_4 \text{ to } x_5, x_6, x_7
 \end{array}$$

$F2$ は x_1, x_2, x_3 から x_4 への接続行列である。 $F3$ は x_4 から x_5, x_6, x_7 への接続行列である。 $F2$ は x_3 から x_4 への出力、 $F3$ は x_4 から x_5 への入力、のための作用要素として理解される。 $F1$ を再利用する全体プログラムの隣接行列が F である。

【0 1 2 8】

【数 2 5】

$$F = \begin{matrix} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{matrix} : \text{entire system matrix}$$

全体プログラムの隣接行列は一巡行列と接続行列とを使って表現される。したがって、単語を基礎とするプログラムの数学的なモデルを使うことが可能である。それによって、手続き型の言語で表現されることができるすべての通常のアigorリズムは、単語に基礎とするプログラミングによって、その構造で実現することができる。

I I I 章 本発明を用いた具体例

<トポロジカルソートの数学的アigorリズム>

具体的方法を示す前に、半順序集合の元を要素とする配列が与えられたとき、それをトポロジカルソートするひとつのアigorリズムを示す。

Lyeeの仕様を単語の系で表現すると、Lyeeにおける仕様は以下のように表現される。

$$x = F(x) \quad (\text{式2-1-3})$$

ここで、汎関数Fは以下で定義されたものである。

$$F(x)(i) = f_i(x(1), x(2), \dots, x(n)), \quad i = 1, 2, \dots, n \quad (\text{式2-1-2})$$

順序集合 (A, \leq) の空でない部分集合Bが \leq の下で全順序集合であるときBを鎖といい、Bのどの2元も比較不能であるときBを反鎖という。1回の反復処理で、右辺の始点が定義されている数以上の端点が定義されるため、Fは単調である。したがってFを繰り返すことによって、次々に生成される反復列

【0129】

【数26】

$$(F^k)_{k \in N}$$

は鎖である。ここで、 k は反復の回数である。最小の反復の回数は最大の鎖のサイズである。一般に、以下の定理によって、半順序集合は、反鎖に分割できる。

[定理]

半順序集合 (A, \leq) の最大の鎖のサイズを n とすると、 A は n 個の互いに素な反鎖に分割される。

この定理は、数学的帰納法によって証明される。この定理を半順序集合である全単語に適用すると、以下のことが言える。すなわち、全単語は、その中では順序がない n 個の単語の集合に分割される。反復列 (数26) の長さ (サイズ) n は単語の始点と端点の関係から求まる鎖のうち最大のもの、すなわち、同期するまでに必要な最小の反復回数を示す。

この定理の証明は、半順序集合の元を要素とする配列が与えられたとき、それをトポロジカルソートするひとつのアルゴリズムを与えている。すなわち、 A_n を反鎖とすると、入力側から、 $A_1, A_2, \dots, A_{n+1}=0$ となるまで次々と求め、 A_n の元を適当な順序で並べ、その後 A_{n-1} の元を適当な順序で並べ、最後に A_1 の元を適当な順序で並べれば、昇順にトポロジカルソートできる。ここで A_n は極大元の集合である。ここでは、すなわち、出力側に一番近い単語の集合である。このように並べた単語は A_1 (入力側) から実行すれば、1度の処理で、すべての単語が次々と成立する。

1. 分割構造を統合する方法

基本構造を1つとし、単語の並べ替えを行う。この方法によって生成されるプログラムは以下の構造を持つ。

- 1) 値の生成のためのW04論理要素はW03へ置く。
- 2) W03論理要素とW04論理要素とを統合した単語とする。等価単語はひとつの単語に統合したものとする。
- 3) スタート画面のW04にすべての基本構造上の単語を置く。このとき、経路の作用要素

についている実行条件はそれぞれの単語に展開する。経路の作用要素は、展開しない。

- 4) すべての単語と入出力作用要素に対して、その始点と基本要素との関係を有向グラフ (Directed Graph) で表現し、上記の方法で、トポロジカルソートして順序を決め配置する。パレット内の反復は不要となるため設置しない。パレット領域のクリアも、不要となり、設置しない。
- 5) Control Areaのフラグ、入出力の作用要素のフラグは、クリアが必要であり、そのまま置く。

この結果、

- 1) 同じ基本構造に属していた単語と入出力作用要素は、その基本構造の経路作用要素についていた実行条件が等しく展開されるので、同一の実行条件を持ち、従って、ソート後もひとまとまりのグループを作り出す。
- 2) 結果的に、経路の実行条件により、基本要素が再グループ化され、そのグループの中では、基本要素は上から実行順に並ぶ。

2. 事例の仕様

この例題の要件は、生徒の成績管理システムの事例である。このシステムでは、総務担当者が生徒の名前を登録し、先生が生徒のテストごとの点数を登録、学期末には総合評価を登録することができる。このシステムは、図11に示す「初期画面」、図12に示す「生徒登録画面」、図13に示す「成績管理画面」の3つの画面と、管理者と先生のIDとパスワードが登録された「ID&パスワードファイル」、生徒の名前やIDが登録されている「生徒名ファイル」、生徒別テストごとの点数が登録されている「テスト別成績ファイル」、生徒ごとの学期末総合評価が登録されている「総合評価ファイル」、の4つのファイルから構成されている。

初期画面は、成績管理システムを起動したときに表示される画面である。Exitボタンを押すとシステムは終了する。IDとパスワードのフィールドに入力して、OKボタンを押すと、システムは、入力されたデータとID&パスワードファイルに登録されているデータを参照（読み込み）し、入力されたデータが総務担当者のものか、先生のものか、あるいは、そのいずれでもないか、を判定する。判定の結果、IDとパスワードが管理者のものならば、生徒登録画面を表示、先生のものなら成績管理画面を表示、いずれのものでもない場合は、初期画面を表示する。

【0130】

生徒登録画面は、総務担当者が生徒名、生徒ID番号を登録する画面である。初期画面で判定したIDとパスワードが、総務担当者のものであった場合に表示される画面で、表示のときには、システムは、生徒名ファイルを参照して、すでに登録済みである生徒のデータを生徒一覧のフィールドに表示する。画面が表示された後、新規登録のフィールドに生徒名と生徒ID番号を入力し、登録ボタンを押すと、入力したデータが生徒名ファ

イルに登録（書き込み）される。戻るボタンを押すと、初期画面に戻ることができる。

【0131】

成績管理画面は、生徒のテストごとの点数と総合評価の登録、参照を行う画面である。初期画面で判定したIDとパスワードが先生のものであったとき表示される。画面表示のときには、システムは、生徒名ファイルを参照して、アクセスしている先生が担当する生徒名と生徒ID番号の一覧を、生徒名のテキストボックスに表示可能な選択候補リストとして用意する。総務担当者が、テキストボックスの右端のボタンを押して、生徒名リストを表示させ、参照したい生徒名とID番号を選択（入力）すると、システムはテスト別成績ファイルを参照して、入力された生徒のテスト結果と、テスト結果の平均点を計算して表示する。総合評価がすでに登録されている場合は、同時に、総合評価ファイルを参照して、総合評価を表示する。新たなテスト結果を登録するときには、テスト結果登録フィールドに、選択されている生徒のデータを入力し、登録ボタンを押す。システムは、新規データをテスト別成績ファイルに登録し、同時に、テスト結果フィールドの内容を新規データも含めた内容に更新して表示し、平均点も再計算して表示する。総合評価を登録するときには、総合評価登録フィールドに、選択されている生徒の総合評価を入力し、登録ボタンを押す。システムは、総合評価のデータを総合評価ファイルに登録し、同時に、総合評価フィールドに登録したデータを表示する。戻るボタンを押すと、初期画面に戻ることができる。

この成績管理プログラムの要件、および、このプログラムを、従来の基本構造に分割した構造にする場合の構造を表わした処理経路図を図14に示す。

1人の学生に対して試験別成績の一覧を表示するような、同じ処理の部分的繰り返しや、平均点の算出のための、集計計算を含んでおり、上に述べた基本的な処理を網羅している。

3. 事例の実装手順

使用言語は、C++、画面部のみC++の機能を用い、残りは、Cの文法のみで記述している。データベース・マネジメントシステムは、Access 97 with DOA である。

設計時、9基本構造に分割した要件を1基本構造のプログラムに統合した形で、実装した。プログラムの生成は、LyeeALL2（Lyee構造プログラムのコード自動生成ツール）を用い、そのテンプレート（基本要素および制御プログラムの雛形）を基本構造統合型Lyeeプログラム用に製作変更して、コード自動生成した。図15は、統合型新プログラムコードの一部である。新プログラムの7行目は、経路の実行条件、すなわちプロセス定義における実行条件であり、基本要素はこれによって再びグループ化している。W02パレットにあった基本要素は、ソート後の順序に並んでいる。単語は、11,12,13,14行のように生成式のみが残る。

基本構造を分割した状態で生成したLyee構造のプログラムと、Lyee方法論による新プログラムと比較すると以下の通りである。この実行ステップ数と時間は、「7人の学生か

らひとりを選び、5試験の成績を表示させる」という要件のプログラムのときのものである。CPUプロセス時間はデータベースのアクセス時間を含まない値、Totalプロセス時間はデータベースのアクセス時間を含む全時間である。処理時間は、プログラムにタイムスタンプの機能を付加し、開始、終了との差として求めた。

4. 結果と評価

1) 結果

【0132】

【表19】

	従来Lyee構造プログラム	本発明適用Lyeeプログラム	
プログラム行数	213,559 行	6,889 行	3.1 %
CPUプロセス時間	150 msec	25 msec	1.5 %
総プロセス時間	470 msec	345 msec	7.3 %

実行コードの減少は主に、単語の構造が小さくなったことと、基本構造が1つになったことによる。主な変化点は、

1. 述語構造はおおよそ21行から成る。これが、通常、代入式1行から成る生成式のみとなる。
2. W03の生成条件とW04の生成式とは1つに統合される。
3. 作用要素については、経路の作用要素は一つのIF文となる。入出力の作用要素はほぼそのまま残る。
4. 80%のクリアの作用要素は取り除かれる。
5. 一つの基本構造は単語や作用要素を除いて約24KBを必要とする。この事例では、9個の基本構造が1つになる。

【0133】

実行ステップ数の減少と実行時間の減少は主に繰り返しが減ったことによる。主な変化点は、

1. すべてのパレットは最低2回繰り返され、基本構造は最低3回繰り返される。これが、1回になる。
2. プログラム・コードは、31%に削減される。
3. しかし全プロセス時間は、それほど減少しない。データベースのアクセス時間が支配的であるためである。
4. データベース処理を用いないリアルタイム処理、たとえばビル監視システムのようなプロセス制御では、CPU時間が支配的である。従って、新方法は、プログラムの大きさ、処理時間とも、プロセス制御の場合に効果大きい。

2) 評価

従来の構造型プログラミングによるプログラムと比較すると、以下のことが、言えるであろう。現時点で、この仕様に基づく従来の構造型プログラミングによるプログラムは、課題を残すものであり、また、従来法はプログラマの技量によるため、定性的比較に留める。

1) プログラムの設計

単語の実行条件は、プロセス定義のアクションの実行条件として、捉えることができ、従来のLyeeにおける設計上の困難が減少する。Lyeeの利点である、宣言的に仕様を捉えることは、新プログラムではプロセス定義と単語の定義で実現される。従って、新プログラムでは、プロセス定義によって制御のための手続きが実現されるため、プログラムの設計は必要ない。従来法では、仕様を基に何らかの設計が必要である。

2) プログラムのサイズ

従来の構造型プログラミングによるプログラムと比較し、新プログラムにおいて、構造上明らかに増加しているコードは、ない。

3) 実行速度

新プログラムでは、繰り返しの回数は必要最小な数に限られるため、従来型プログラムに比べて実行速度が遅くなる理由はない。

I V 章 むすび

現状では、Lyee方法論に基づくツールによって生成されるプログラムは、設計時点で分割した仕様を、分割した状態でプログラムモジュールに展開したものである。確かに、要件に明白に表されている順序に関する情報を設計に活用することは、設計の精度をあげ、効率化に役立つ。しかし、Lyee構造のプログラムは、実行前に、静的に解決できる問題点を、実行時点で動的に解決しているため、コードのサイズは大きく、処理速度も遅い。ここに改善の余地がある。設計の効率化の効果を生かしつつ、プログラムの意味を変えずに、初めから基本構造に分割せずに、プログラムを生成することが出来る。そうすることによって、単語全体は1つの領域となる。したがって、単語の関係が作り出す有向グラフ上のトポロジカルソートで、単語全体の正しい順序が決定されると、順序性を確保するための繰り返しは不要となる（特許文献6参照）。

したがって、処理経路図を作成すること、すなわち要件を分割してとらえることは、システムの要件を正確にとらえるために非常に有用である。処理経路図を作成することは、出力条件に共通要素を持つ単語を論理体の単位でとらえることである。条件の共通要素を経路作用要素に割り当てることは、効率的にユーザ要件を定義するためには大きなメリットがある。

【0134】

一方、プログラムのコードを生成し、実行する時点では、基本構造に分割したシステムを統合することが有用である。なぜなら、システム全体の単語単位のプログラムの実行順序を、最少実行回数で出力単語の生成を完了できる順序に並べることができるから

である。

【0135】

本明細書では、単語単位の宣言実行モジュールからなるシステムにおける、出力データを生成するための宣言実行モジュールの処理を、無駄な繰り返しを排除して最少実行回数で完了し、かつ、プログラムサイズを小さくすることを課題として、以下の手段で解決することについて述べた。

- 1) 同じ出力条件を持つ単語の集合（論理体）を、1つの基本構造として、システム全体を処理経路図でとらえる。これによって、同じ基本構造に属する宣言実行に共通する実行条件（経路作用要素がその基本構造を指定する条件）を的確にとらえることができる。
- 2) 経路作用要素もトポロジカルソートの対象とし、単語の関係を定義している論理要素と同じく、その要件を単語の関係で定義する。すなわち、どのような条件が成立したとき（定義式の実行条件）、どの基本構造の実行を指定するか（定義式）、を定義する。
- 3) この経路作用要素の定義式実行条件を、その条件が成立するときに経路作用要素が指定する基本構造に属するモジュールの実行条件に加えることによって、経路作用要素を排除する。経路作用要素の実行条件を、指定先の基本構造のモジュールの実行条件に置くことによって、経路作用要素を削除しても、基本構造間のリンク（すなわち単語間のリンク）が成立するので、プログラム全体を1つにまとめることが可能になる。
- 4) こうして得られた単一化されたプログラム全体の宣言実行モジュール群（経路作用要素を含めない）に対して、トポロジカル・ソートを行って宣言実行モジュール群を最適順序に並び替える。これにより、無駄な繰り返しを回避し、最少実行回数でプログラムを実行することが可能となる。

【0136】

なお、本発明は、上述した実施形態および実施例には限定されず、本発明の技術思想の範囲内で様々な変形が可能である。たとえば、ビジネス・メソッド、ソフトウェア開発装置、ソフトウェア開発支援装置、ソフトウェア開発管理装置、あるいはこれらの機能をコンピュータに実現するためのソフトウェア並びに該ソフトウェアを搭載した記録媒体・専用機、などとしてもそれぞれ実現することが可能である。さらに、上記で説明したように本発明は、方法としても、或いはかかる機能を備えるソフトウェアとしても、該ソフトウェアが搭載される装置・ツール（ソフトウェア自体の場合も含む）としても、さらにはシステムとしても、実現され得るのはもとよりである。

【0137】

例えば図16は、本発明の異なる一実施形態として、「開発対象のソフトウェア」を生産するためのプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、或いはソフトウェア開発管理装置のいずれかとして本発明を実施する場合に機能として備える構成を示した機能ブロック図である。

【0138】

全体制御部1601は、本プログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、或いはソフトウェア開発管理装置の全体の動作制御、タイミング制御、入出力制御等を行う機能を有しており、当該機能を持つ専用チップ、専用回路、または当該機能をコンピュータに果たさせるためのソフトウェア（ツールとしてのソフトウェアも含む）、或いは該ソフトウェアを記録した記録媒体、当該記録媒体を搭載した処理装置・管理装置・ツールとして実現される。

【0139】

図17は、上記の構成を備えるプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、或いはソフトウェア開発管理装置のいずれかとして実施される本発明の動作を示したフローチャートである。

【0140】

両図の説明は省略する。

【0141】

上記のような構成を備える本発明によれば、単語単位の宣言実行モジュールからなるプログラムにおける、出力データを生成するための宣言実行モジュールの処理を、無駄な繰り返しを排除して最少実行回数で完了し、かつ、プログラムサイズを小さくすることが可能となる。

【0142】

本発明の異なる実施体としての「開発対象のソフトウェア」を生産するためのプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、ソフトウェア開発管理装置は、1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から、要件充足に必要な全ての、L2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）のいずれかの宣言の実行単位を規定する手段と、前記に既定された、全てのL2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）の（半）順序関係を定義する手段と、前記ステップによって定義された（半）順序関係で規定される前記L2処理、L処理、I2処理、O4処理に対してトポロジカル・ソートを行う手段と、前記第3のステップによって並び替えられた宣言の実行単位の順序列にしたがって該宣言実行単位に該当するL y e e

方法論に基づく所与のコード列を配置する手段とを具備するように構成することもできる。

【0143】

本発明はさらに、上述の「開発対象のソフトウェアを生産する方法」によって生産されたソフトウェア、及び当該ソフトウェアが搭載された記録媒体或いは当該ソフトウェアが搭載された装置（ハードウェア）としても実現されるが、この場合の本発明は、1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から規定された、要件充足に必要な全ての、L2処理（入力単語の属性チェック処理）、L処理（出力単語の値生成処理）、I2処理（論理体入力処理）、O4処理（論理体出力処理）の宣言の実行単位が、単語単位に宣言された要件から定義した（半）順序関係に基づいて行われるトポロジカル・ソートによって並び替えられた順序列にしたがって、該当するL y e e方法論に基づく所与のコード列として構成されることもできる。

【0144】

またさらに本発明は、上述の「開発対象のソフトウェアを生産する方法」によってソフトウェアを生産するために用いられるソフトウェアコードの雛型としてのソフトウェア、及び当該ソフトウェアが搭載された記録媒体或いは当該ソフトウェアが搭載された装置（ハードウェア）としても実現される。

【0145】

さらに、本発明は、上述の「開発対象のソフトウェアを生産する方法」による、要件から抽出した情報（ドキュメント（紙、データ））の抽出方法として、またかかる抽出方法によって抽出された情報（ドキュメント（紙、データ））として、さらには当該抽出された情報の使用方法として、或いは、これらの情報が搭載された情報記録媒体として、または情報の抽出方法／使用方法がコード化されたソフトウェア、当該ソフトウェアが搭載された記録媒体／装置（ハードウェア）として、いずれも実現することができる関連づける情報とを備えるソフトウェア開発要件から抽出した情報として実現してもよい。

【0146】

本発明は当該技術分野における通常の知識を有する者にとって修正および改変が容易に数多くなし得るので、図示および記述されたものと寸分違わぬ構成および動作に本発明を限定することは望ましくないことであり、従って、あらゆる適切な改変体および等価体は本発明の範囲に含まれるものと見なされうる。前述の本発明に係る実施の具体的な形態の説明および例示によって詳細に記述されたが、本願の特許請求の範囲のみならず本発明に係る開示事項全体に定義された本発明の範囲から逸脱することなしに、修正、置換、および、変更が数多く可能である。

【0147】

また、本願に係る発明は、その適用において、上記の記述において説明されるか、或いは、図面に示された要素の詳細な解釈及び組み合わせに限定されるものではない。本発明は、他の実施形態が可能であり、種々の方法で実用および実施可能である。また、ここで用いられた語法および用語は記述を目的とするものであり、限定的に働くものとみなされてはならない。

【0148】

従って、当該技術分野における通常の知識を有する者は、本開示の基調となる概念は、本発明の幾つかの目的を実施するための他の構造、方法、及び、システムを設計するための基礎として容易に利用され得ることを理解するはずである。従って、本発明の趣旨および範囲から逸脱しない限り、本願の特許請求の範囲にはそのような等価な解釈が含まれるものと見なされるものである。

【0149】

上記の詳細な説明では、有向グラフ表記、隣接行列算出、トポロジカル・ソート、並び替えという順序で本発明の思想を実現する方法について述べた。しかしこれらはどれも絶対不可欠な要素というものではなく、例えば、有向グラフ表記の表記を経ないで直接隣接行列算出→トポロジカル・ソート→並び替えという順序で本発明の思想を実現してもよい。

【0150】

さらに、有向グラフに対して、例えば幅優先探索 (Breadth-First Search)、反復深化法 (Iterative Deeping)、発見的探索 (Heuristic Search)、ヒル・クライミング (Hill Climbing)、最適優先探索 (Best-First Search)、オイラー (Euler) の一筆書き、ダイクストラ (Dijkstra) 法等の各種探索技法を組み合わせることで並び替えてもよい。

【0151】

或いは、隣接行列算出のあとを、これらの各種探索技法を組み合わせることで本発明の根本思想を実現してもよい。

【0152】

また、本発明に係る技術思想は、例えばコンピュータソフトウェアの自動開発装置、自動開発プログラム、自動開発プログラムを記録した記録媒体、伝送媒体、紙媒体としても、また、自動開発プログラムを登載したコンピュータ・装置、自動開発プログラムを実行するクライアント・サーバ形式等といったカテゴリーにおいても実現、利用可能であることはいうまでもない。

【0153】

さらに本発明は、単一プロセッサ、単一ハードディスクドライブ、及び、単一ローカルメモリを備えたコンピュータシステムに限らず、当該システムのオプションとして、任意の複数または組み合わせプロセッサ又は記憶デバイスを装備するにも適している。コンピュータシステムは、精巧な計算器、掌タイプコンピュータ、ラップトップ／ノートブックコンピュータ、ミニコンピュータ、メインフレームコンピュータ、及び、スーパーコンピュータ、ならびに、これらの処理システムネットワーク組合わせを含む。本発明の原理に従って作動する任意の適切な処理システムによって代替されうるし、また、これらと組合せて用いることも可能である。

【0154】

また、本発明に係る技術思想は、もとよりあらゆる種類のプログラミング言語に対応可能である。さらに本発明に係る技術思想は、あらゆる種類・機能のアプリケーションソフトウェアに対しても適用可能である。

【0155】

またさらに本願発明は、その技術思想の同一及び等価に及ぶ範囲において様々な変形、追加、置換、拡大、縮小等を許容するものである。また、本願発明を用いて生産されるソフトウェアが、その2次的生産品に登載されて商品化された場合であっても、本願発明の価値は何ら減ずるものではない。

【産業上の利用可能性】

【0156】

上記で規定される本発明では、繰返しをなくす前処理を自動で行い、こうして得られた前処理済の単語単位のプログラムをもとにL y e e（登録商標）方法論により自動的に目的プログラム生成を行う。つまり、ユーザ要件の洗練化（整列）から目的プログラムの生成に至るまで自動化することが可能となる。これにより、ソフトウェア生産の大幅な効率向上、生産性向上、品質向上等、ソフトウェア産業上に大きな効果をもたらす。

【図面の簡単な説明】

【0157】

【図1】 本発明の一実施形態に係るセルの概念を説明するための概念図である。

【図2】 本発明の一実施形態に係る例1のシステムの画面の説明図である

【図3】 本発明の一実施形態に係る例1のシステムの単語を基本構図単位に示した説明図である。

【図4】 本発明の一実施形態に係る例1のシステムの単語の関係を表わした有向グ

ラフである。

【図 5】本発明の一実施形態に係る例 1 のシステム全体を 1 つの隣接行列 F で表わす過程を説明するための図である。

【図 6】本発明の一実施形態に係る隣接行列 F がトポロジカルソート済みであることを説明するための図である。

【図 7】本発明の一実施形態に係る例 1 のシステムの構造から経路作用要素を削除したときの、単語の関係を表わした有向グラフである。

【図 8】本発明の一実施形態に係る例 1 のシステムの構造から経路作用要素を削除したとき、システム全体を 1 つの隣接行列 F' で表わす過程を説明するための図である。

【図 9】本発明の一実施形態に係る隣接行列 F' がトポロジカルソート済みであることを説明するための図である。

【図 10】本発明の一実施形態に係る例 2 を有向グラフである。

【図 11】本発明の一実施形態に係る例 2 の「初期画面」を表した図である。

【図 12】本発明の一実施形態に係る例 2 の「生徒登録画面」を表した図である。

【図 13】本発明の一実施形態に係る例 2 の「成績管理画面」を表した図である。

【図 14】本発明の一実施形態に係る例 2 の成績管理プログラムの要件、および、このプログラムを、従来の基本構造に分割した構造にする場合の構造を表わした処理経路図である。

【図 15】本発明の一実施形態に係る統合型新プログラムコードの一部である。

【図 16】本発明の異なる一実施形態として、「開発対象のソフトウェア」を生産するためのプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、或いはソフトウェア開発管理装置のいずれかとして本発明を実施する場合に機能として備える構成を示した機能ブロック図である。

【図 17】本発明の一実施形態に係る上記の構成を備えるプログラム（ソフトウェア）、プログラム生成装置、プログラム処理装置、ツール（装置として或いはソフトウェアとしての双方を含む）、ソフトウェア開発装置、ソフトウェア開発支援装置、或いはソフトウェア開発管理装置のいずれかとして実施される本発明の動作を示したフローチャートである。

【符号の説明】

【0158】

- 1 0 1 同期範圍
- 1 1 0 1 基本構造B S 1 (入力)
- 1 1 0 2 基本構造B S 2
- 1 1 0 3 基本構造B S 3
- 1 1 0 4 基本構造B S 1 (出力)
- 1 2 0 1 隣接行列F 1'
- 1 2 0 2 隣接行列F 2'
- 1 2 0 3 隣接行列F 3'
- 1 2 0 4 結合隣接行列F C 1'
- 1 2 0 5 結合隣接行列F C 2'

【書類名】 請求の範囲**【請求項 1】**

1つのプログラムとして実装するユーザ要件を、アクセス条件を伴う論理体ごとに、該論理体上の単語ごとに、単語名、定義式、該定義式の実行条件、入出力属性、単語の値の属性によって宣言された単語単位の宣言から、要件充足に必要な全ての、L 2 処理（入力単語の属性チェック処理）、L 処理（出力単語の値生成処理）、I 2 処理（論理体入力処理）、O 4 処理（論理体出力処理）のいずれかの宣言の実行単位を規定する第 1 のステップと、

前記に規定された、全ての L 2 処理（入力単語の属性チェック処理）、L 処理（出力単語の値生成処理）、I 2 処理（論理体入力処理）、O 4 処理（論理体出力処理）の（半）順序関係を定義する第 2 のステップと、

前記ステップによって定義された（半）順序関係で規定される前記 L 2 処理、L 処理、I 2 処理、O 4 処理に対してトポロジカル・ソートを行う第 3 のステップと、

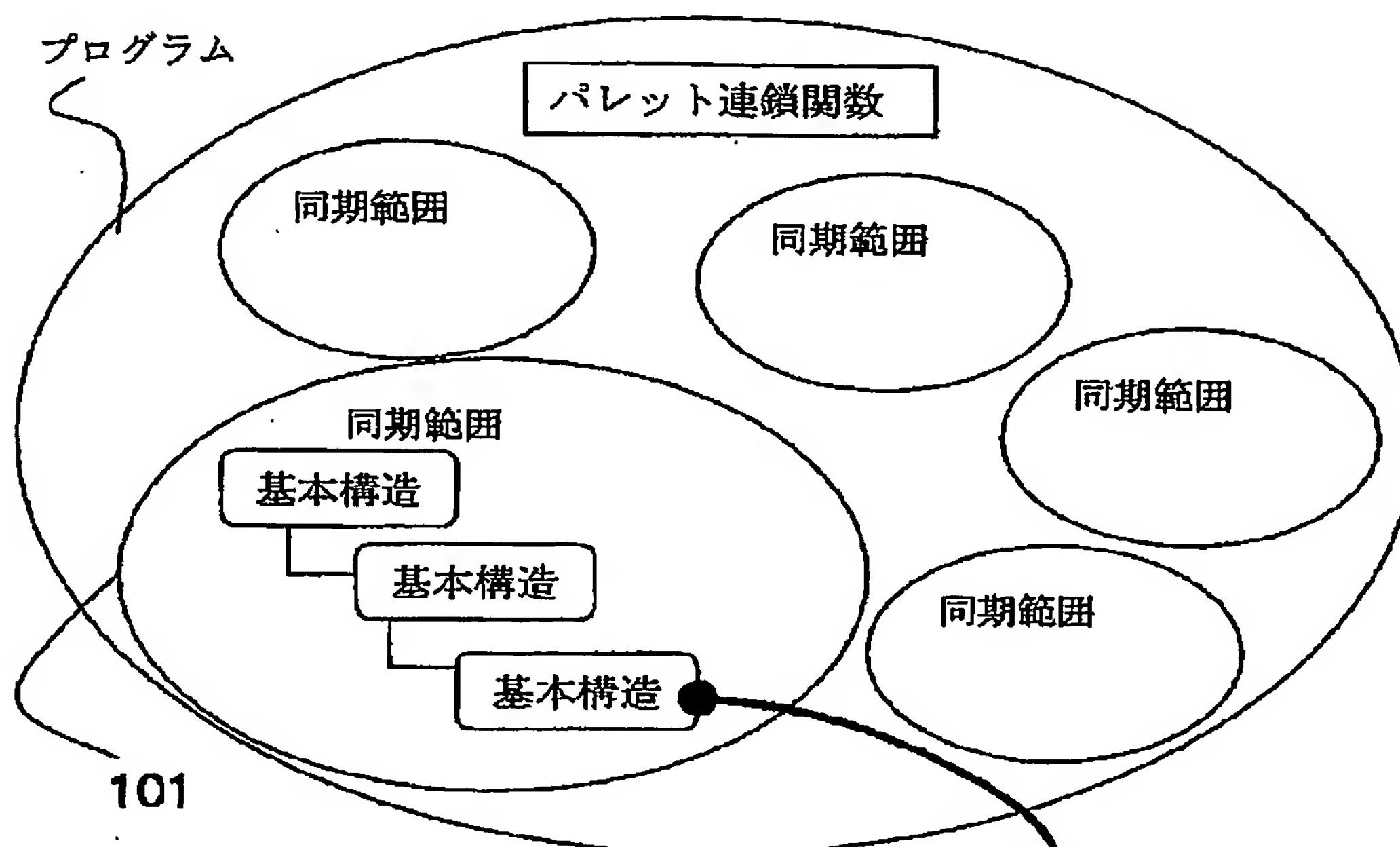
前記第 3 のステップによって並び替えられた宣言の実行単位の順序列にしたがって該宣言実行単位に該当する L y e e 方法論に基づく所与のコード列を配置する第 4 のステップと

を具備することを特徴とするソフトウェア生成方法。

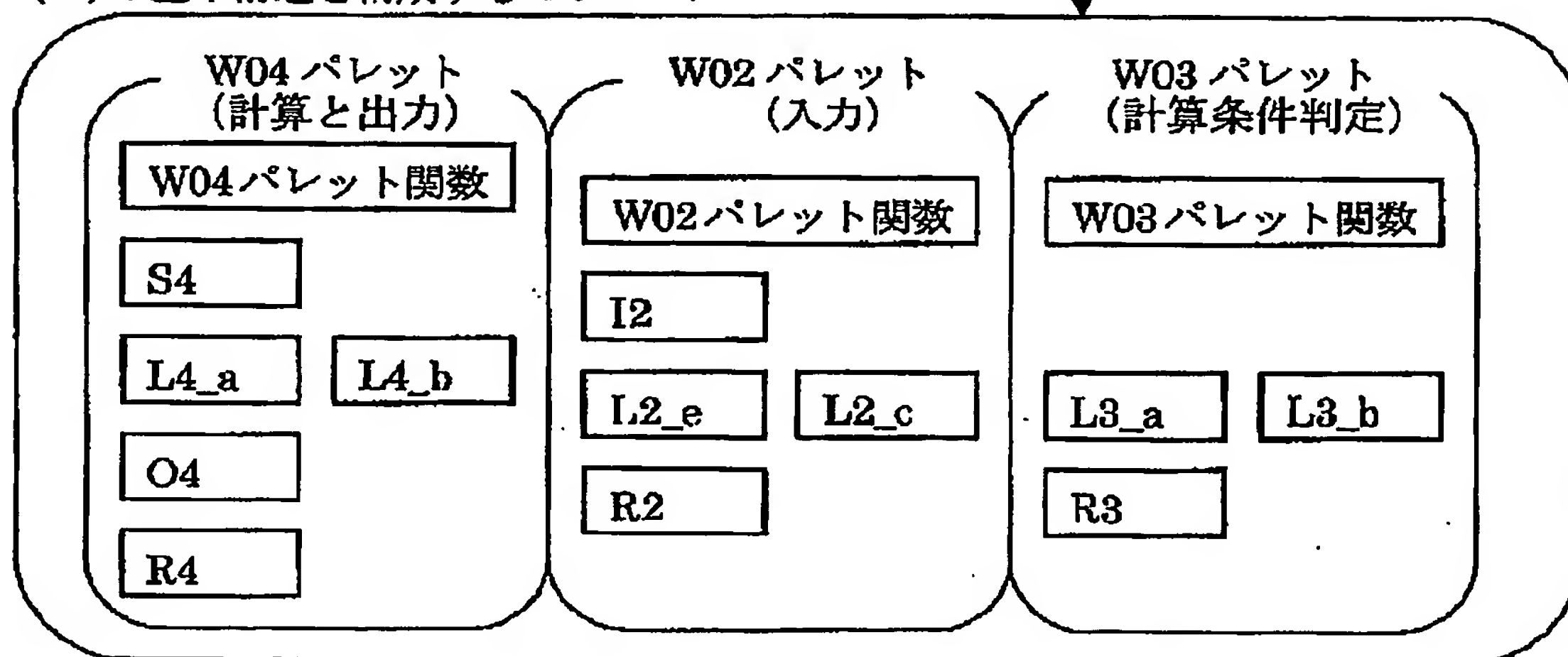
【書類名】 図面

【図 1】

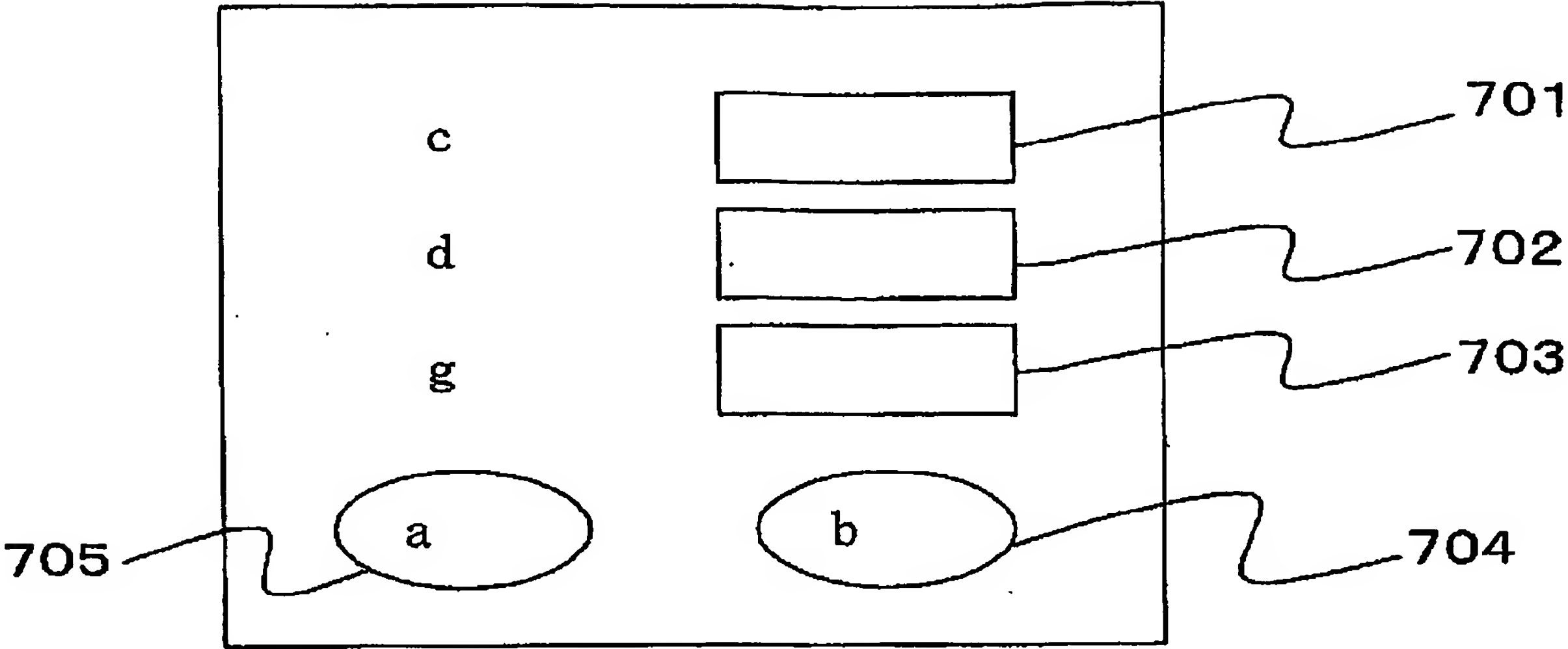
(a) : L y e eプログラムの分割単位



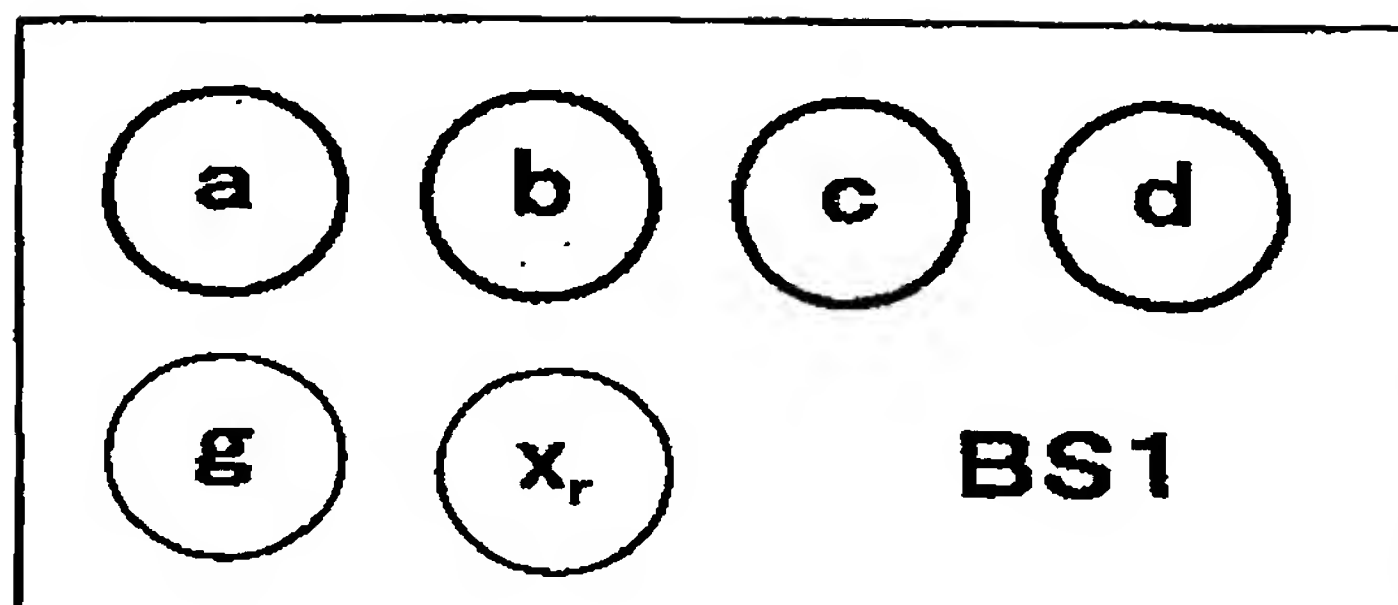
(b) : 基本構造を構成するモジュール



【図 2】

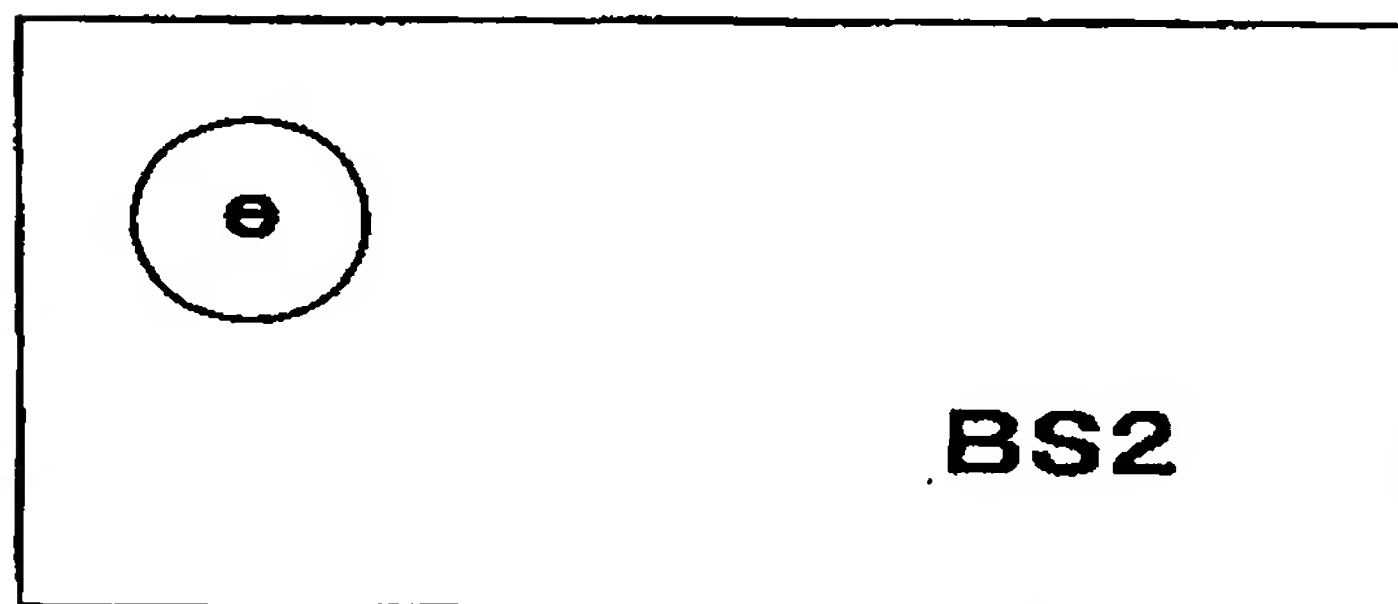


【図 3】

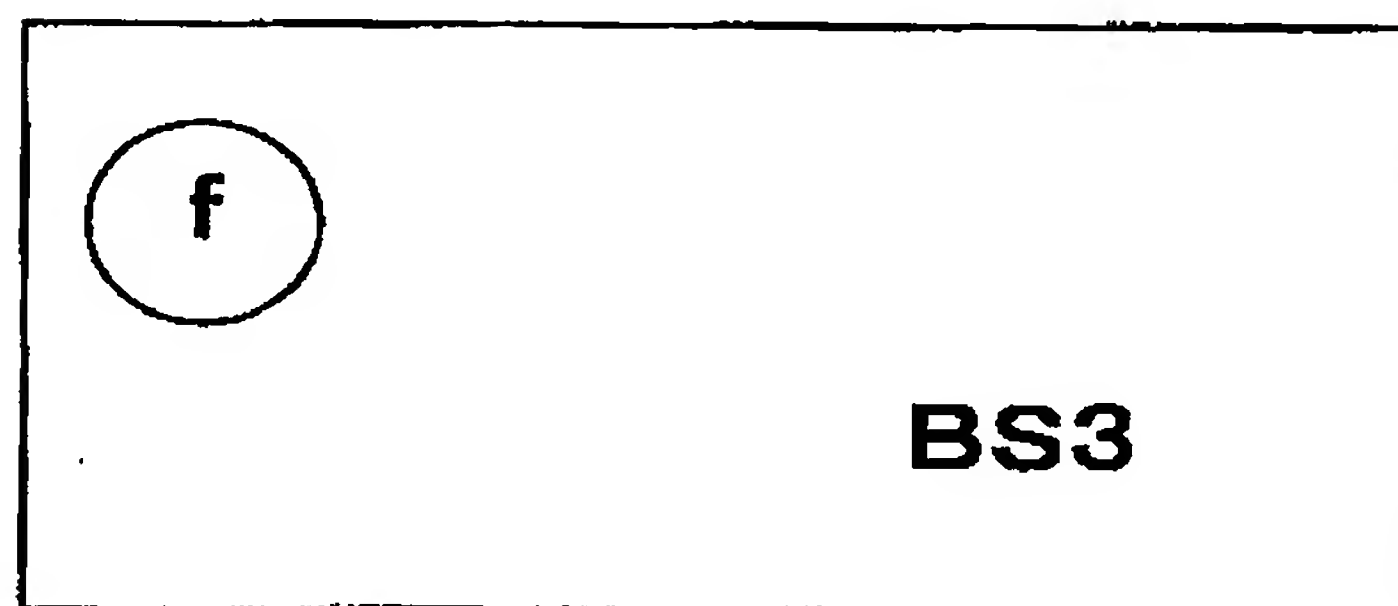


$a = \text{真のとき } x_r = \text{BS2}$
 $b = \text{真のとき } x_r = \text{BS3}$

$e = \text{真のとき } g = e$
 $f = \text{真のとき } g = f$

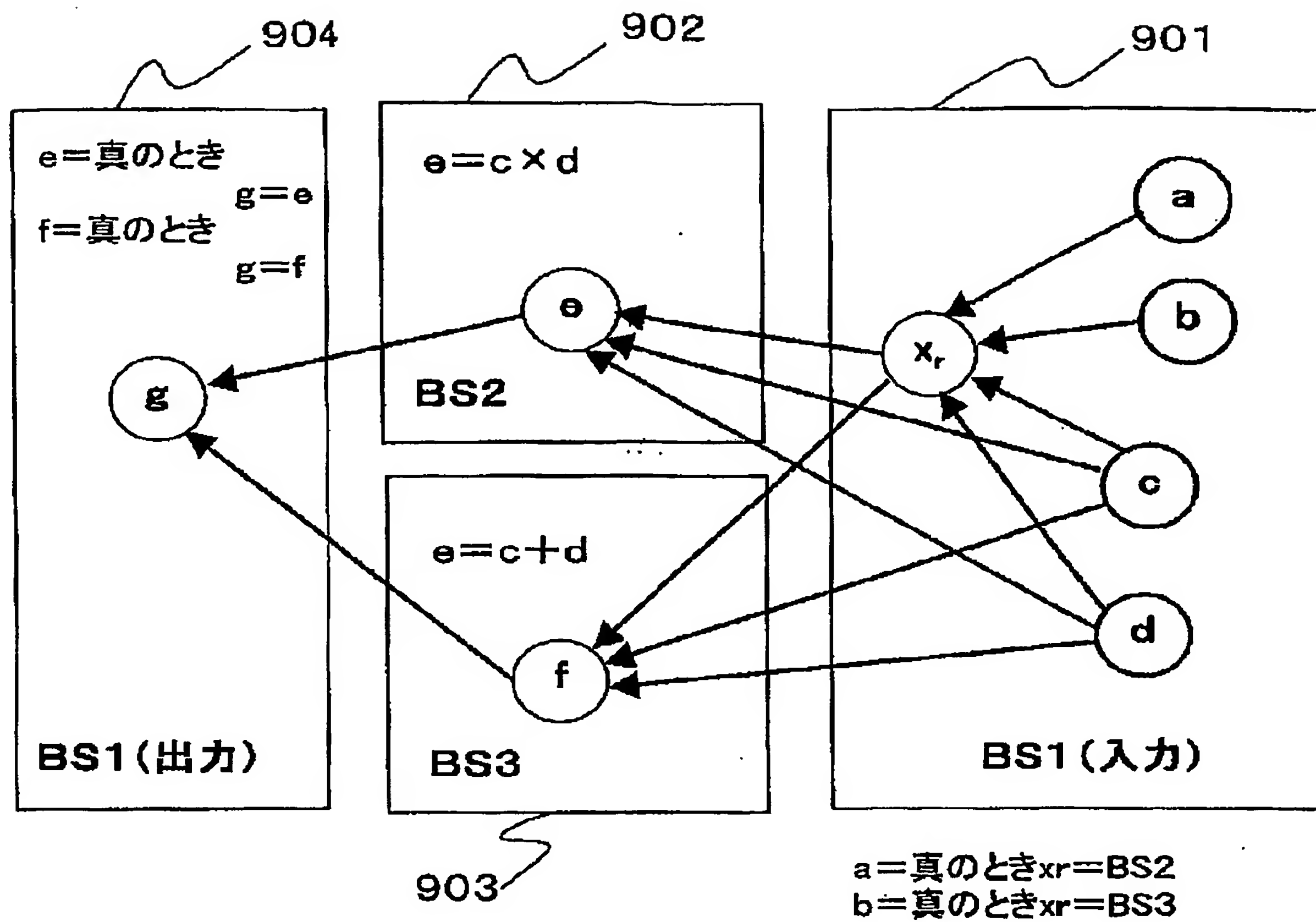


$e = c + d$



$f = c \times d$

【図 4】



【図 5】

The diagram shows a 7x7 matrix with rows and columns labeled *a*, *b*, *c*, *d*, *x_r*, *e*, *f*, and *g*. The matrix is partitioned into several regions indicated by callouts:

- 1001**: A 5x5 region covering rows *a* to *x_r* and columns *a* to *x_r*.
- 1006**: A 2x2 region covering rows *e* and *f* and columns *e* and *f*.
- 1007**: A 5x3 region covering rows *a* to *x_r* and columns *e* to *g*.
- 1002**: A 2x3 region covering rows *e* and *f* and columns *e* to *g*.
- 1008**: A 1x3 region covering row *g* and columns *e* to *g*.
- 1003**: A 1x1 region covering row *g* and column *g*.
- 1004**: A 2x5 region covering rows *e* and *f* and columns *a* to *x_r*.
- 1005**: A 1x5 region covering row *g* and columns *a* to *x_r*.

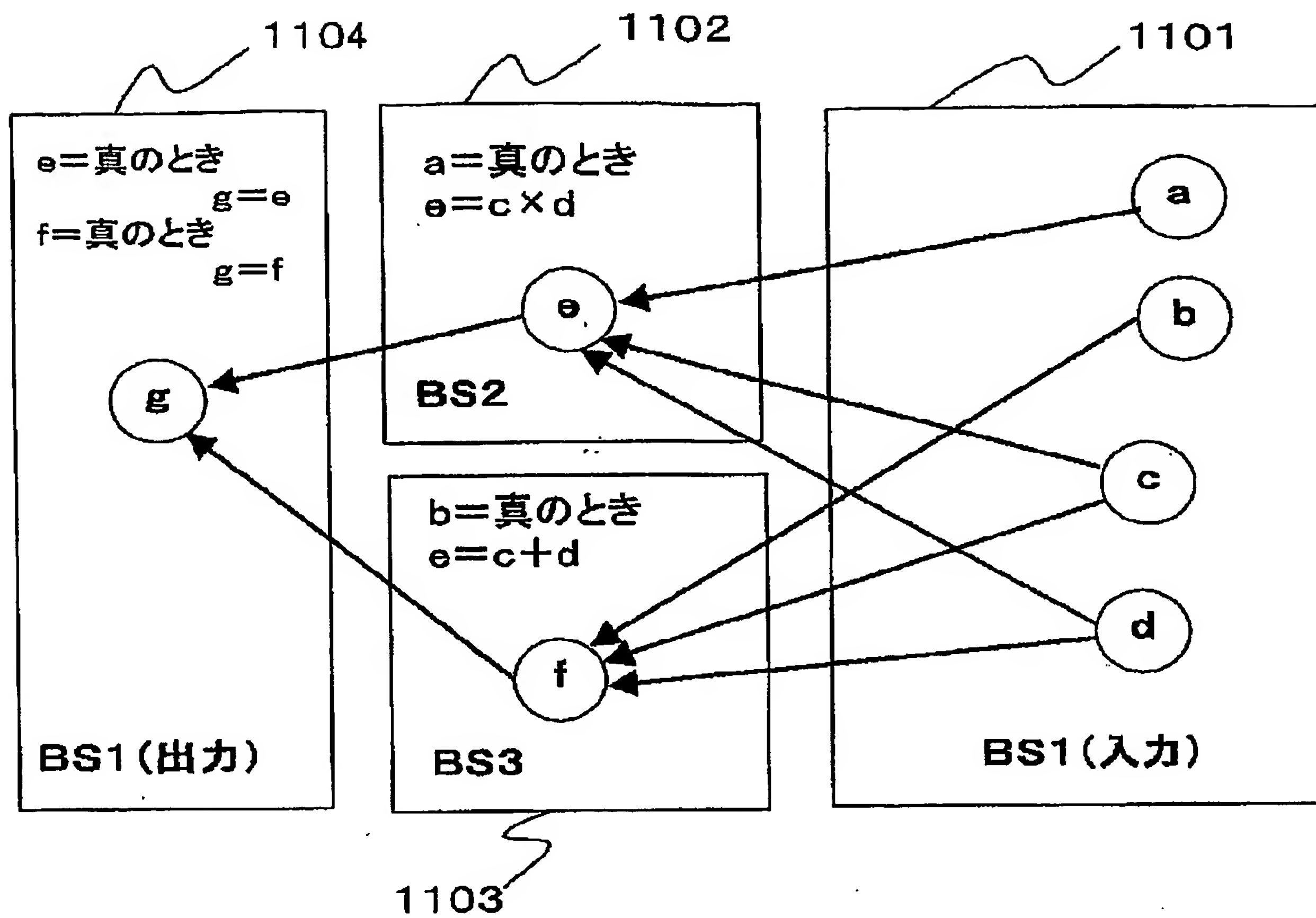
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>x_r</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	0	0	0	0	0	0	0
<i>b</i>	0	0	0	0	0	0	0	0
<i>c</i>	0	0	0	0	0	0	0	0
<i>d</i>	0	0	0	0	0	0	0	0
<i>x_r</i>	1	1	1	1	0	0	0	0
<i>e</i>	0	0	1	1	1	0	0	0
<i>f</i>	0	0	1	1	1	0	0	0
<i>g</i>	0	0	0	0	0	1	1	0

【図 6】

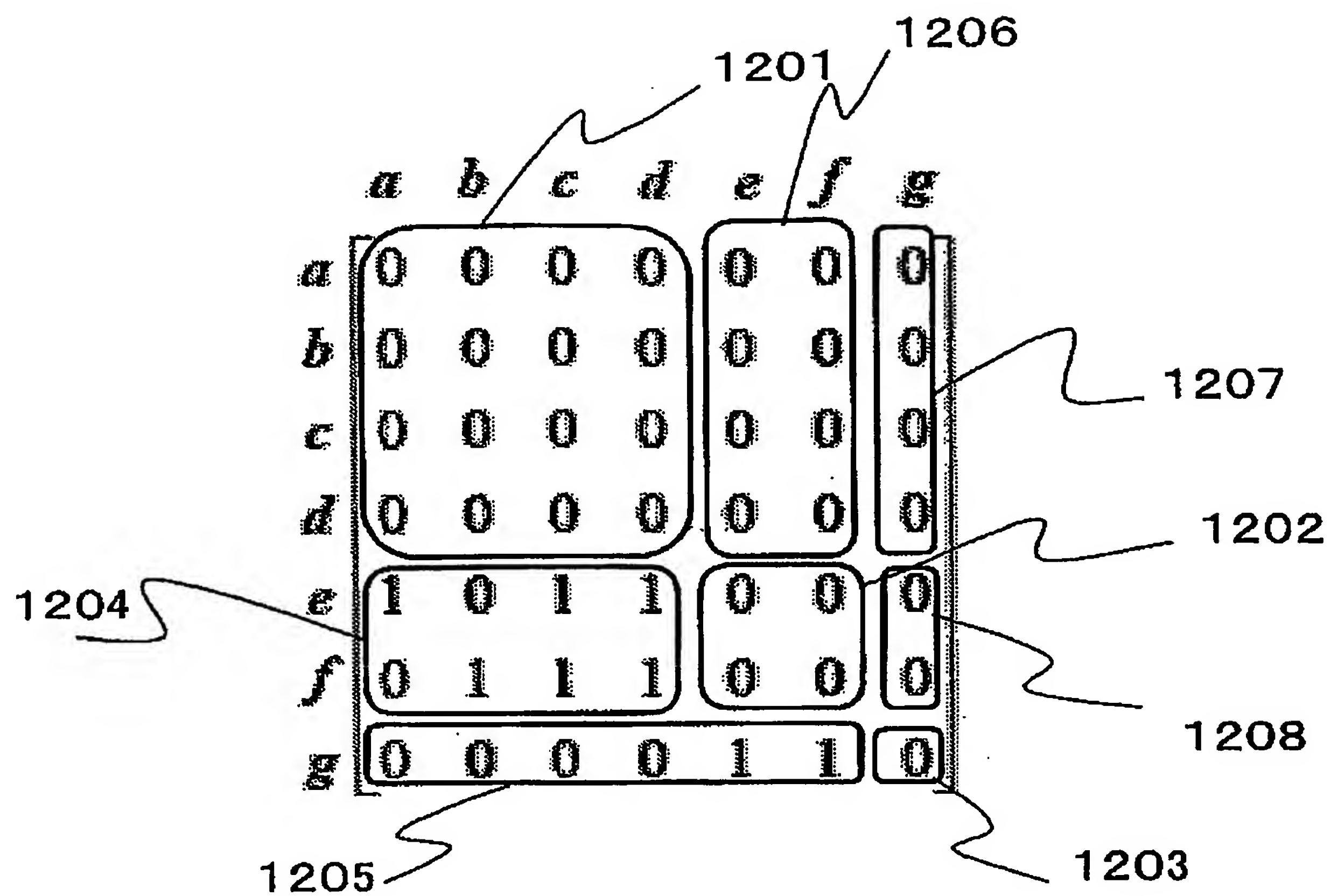
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>x_r</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	0	0	0	0	0	0	0
<i>b</i>	0	0	0	0	0	0	0	0
<i>c</i>	0	0	0	0	0	0	0	0
<i>d</i>	0	0	0	0	0	0	0	0
<i>x_r</i>	1	1	1	1	0	0	0	0
<i>e</i>	0	0	1	1	1	0	0	0
<i>f</i>	0	0	1	1	1	0	0	0
<i>g</i>	0	0	0	0	0	1	1	0

1 3 0 1

【図 7】



【図8】

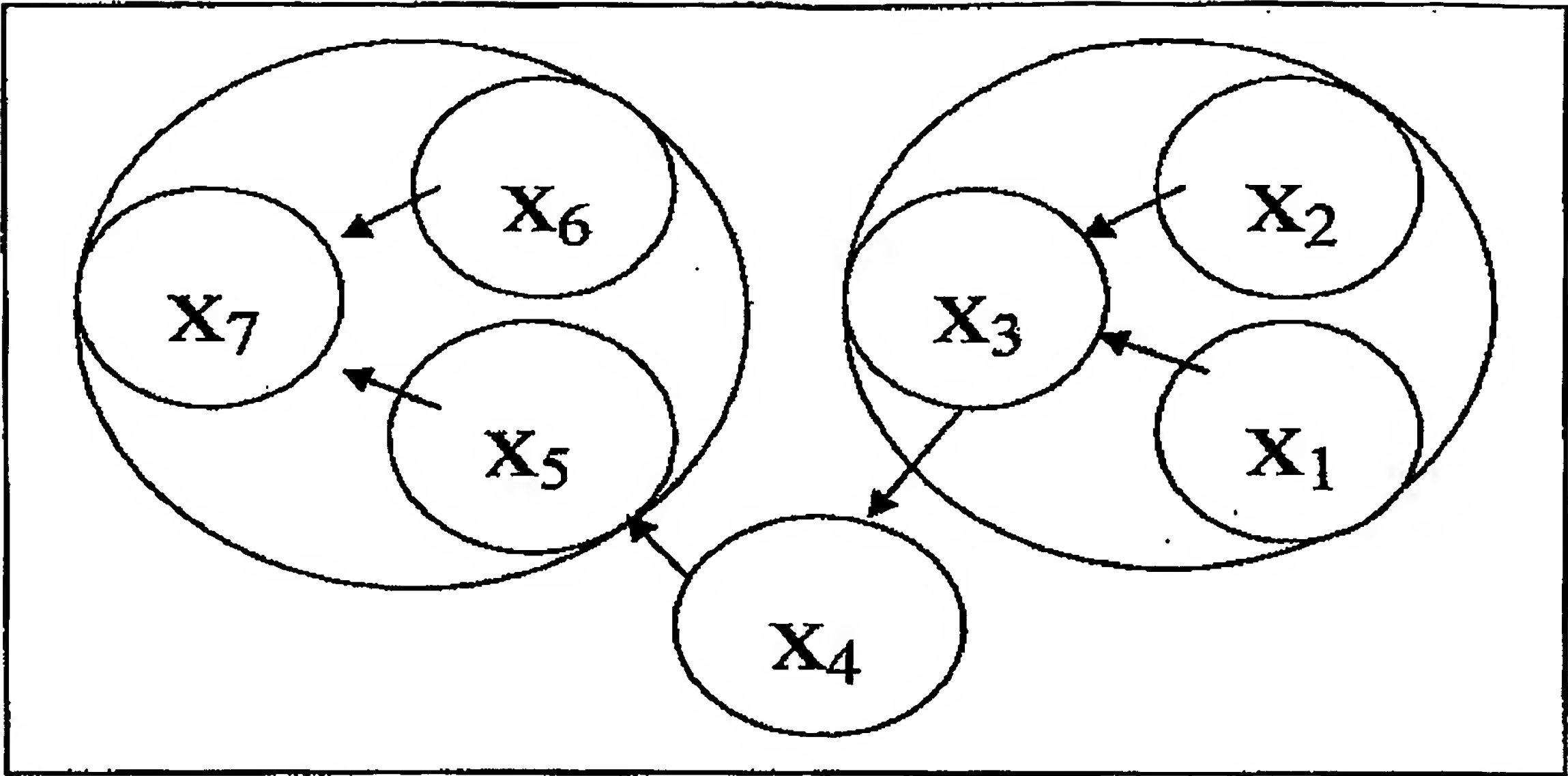


【図 9】

	a	b	c	d	e	f	g
a	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
e	1	0	1	1	0	0	0
f	0	1	1	1	0	0	0
g	0	0	0	0	1	1	0

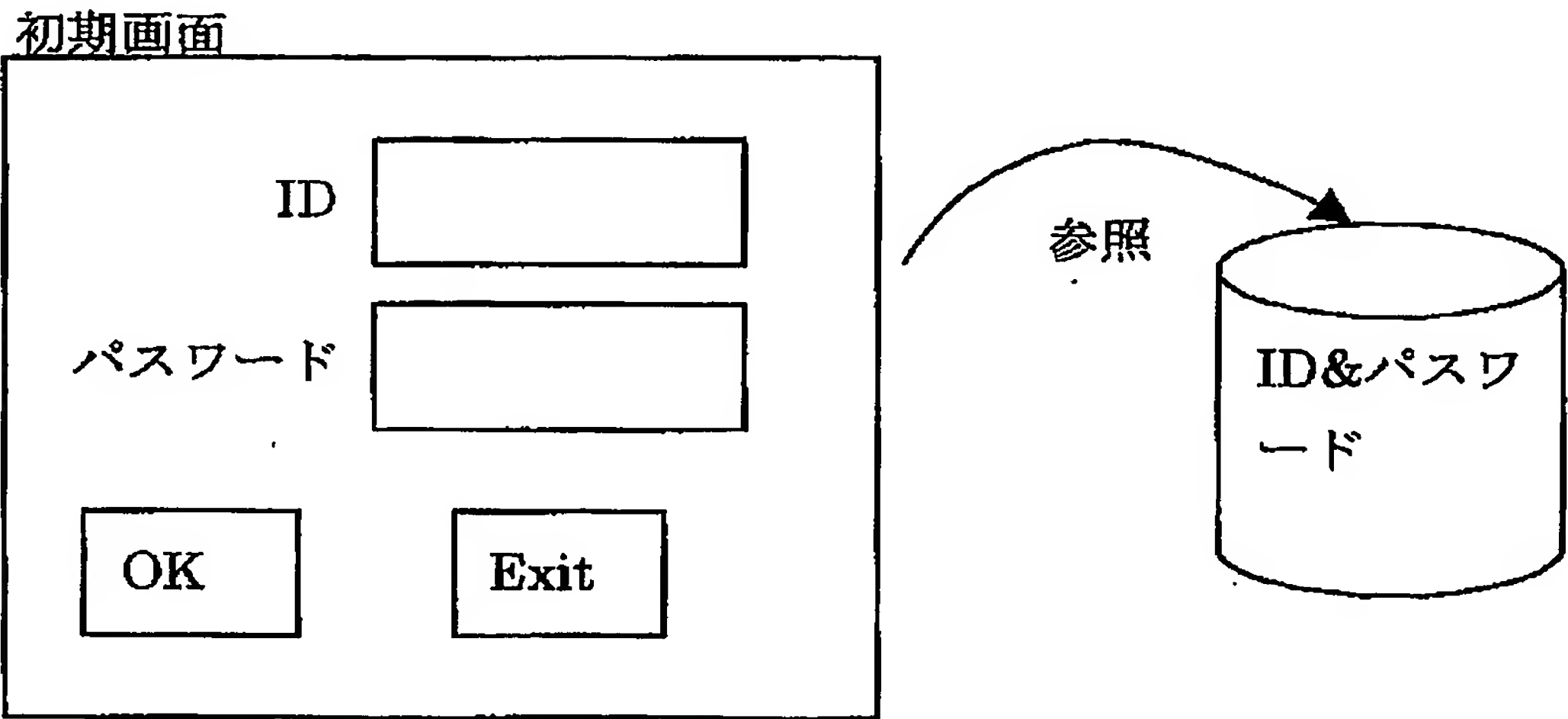
1 6 0 1

【図 10】

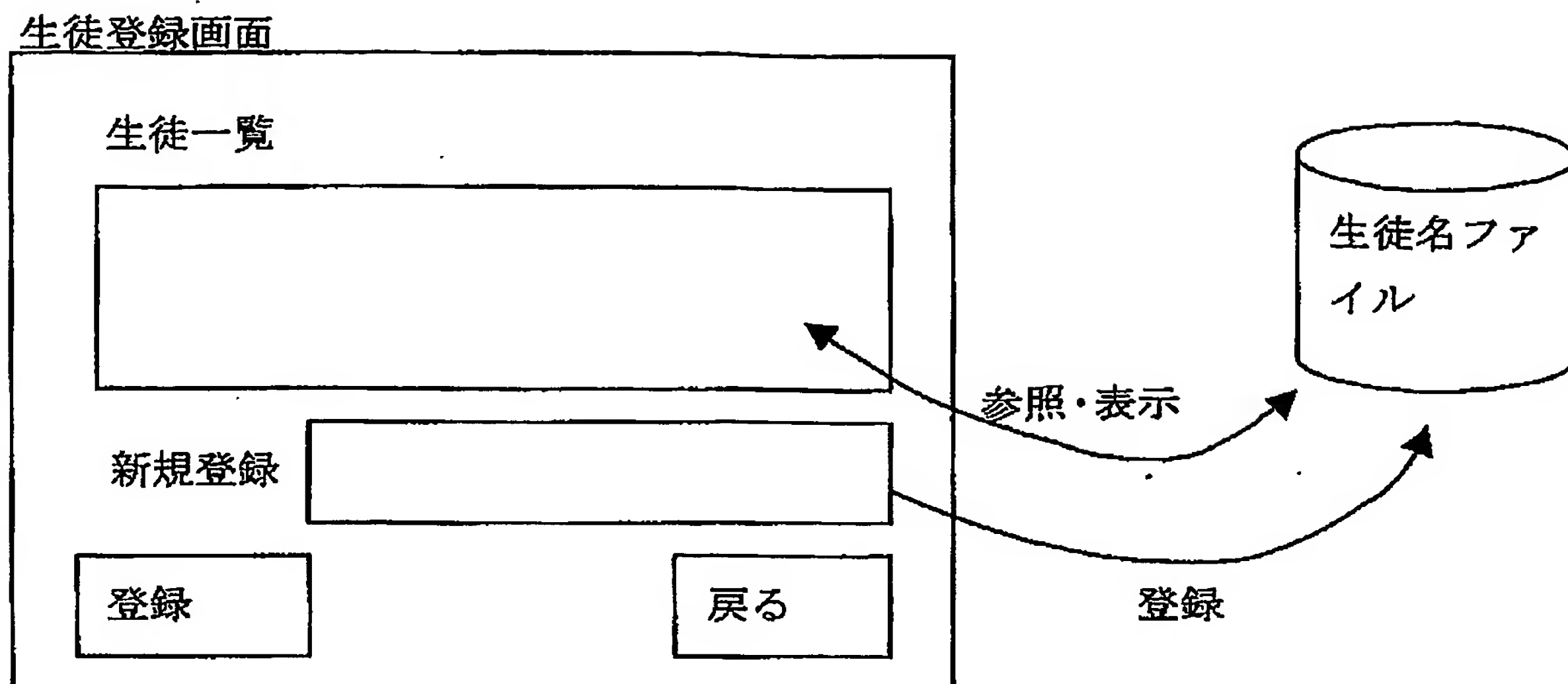


集計の有効グラフ

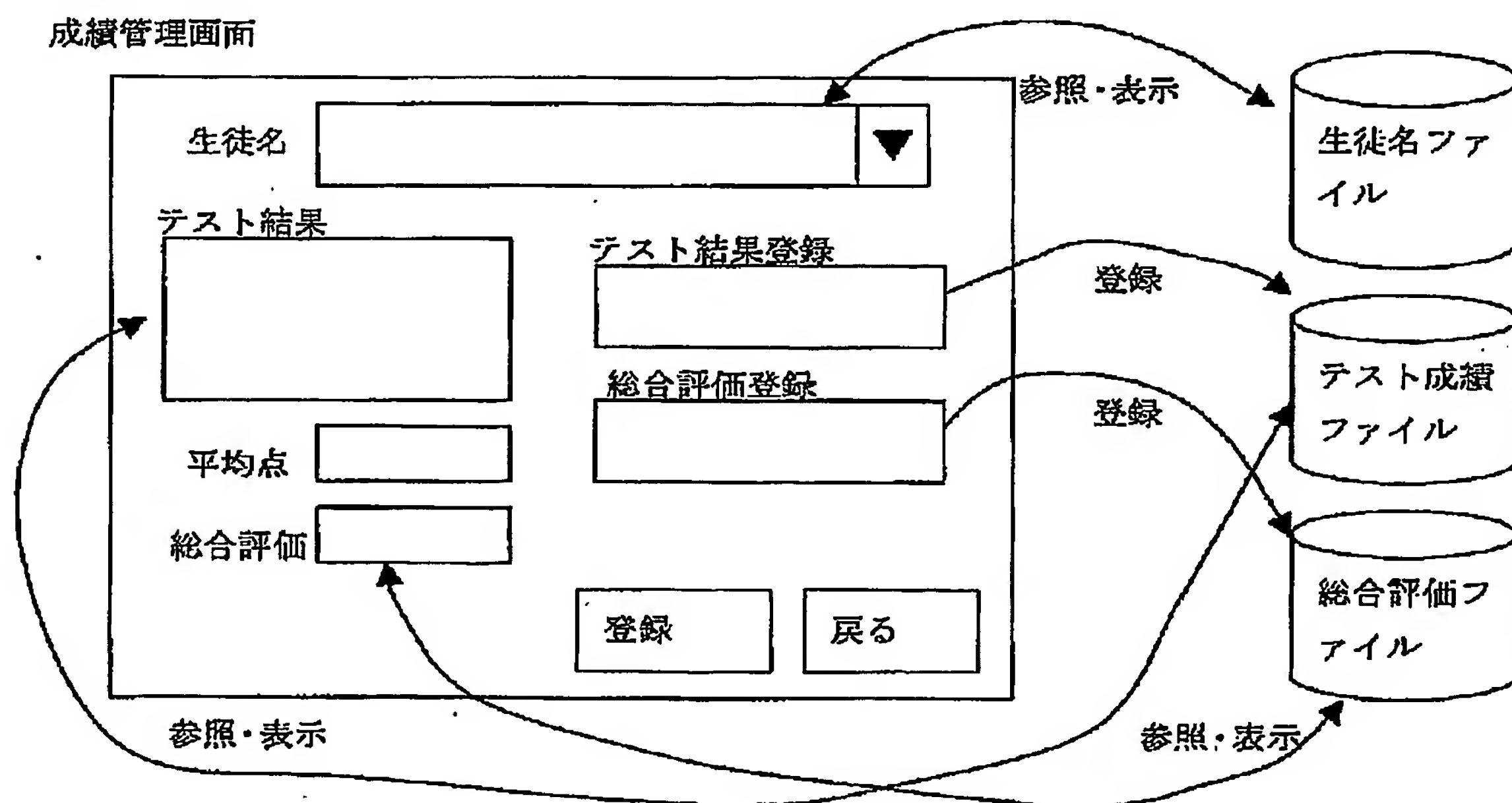
【図 1 1】



【図 1 2】

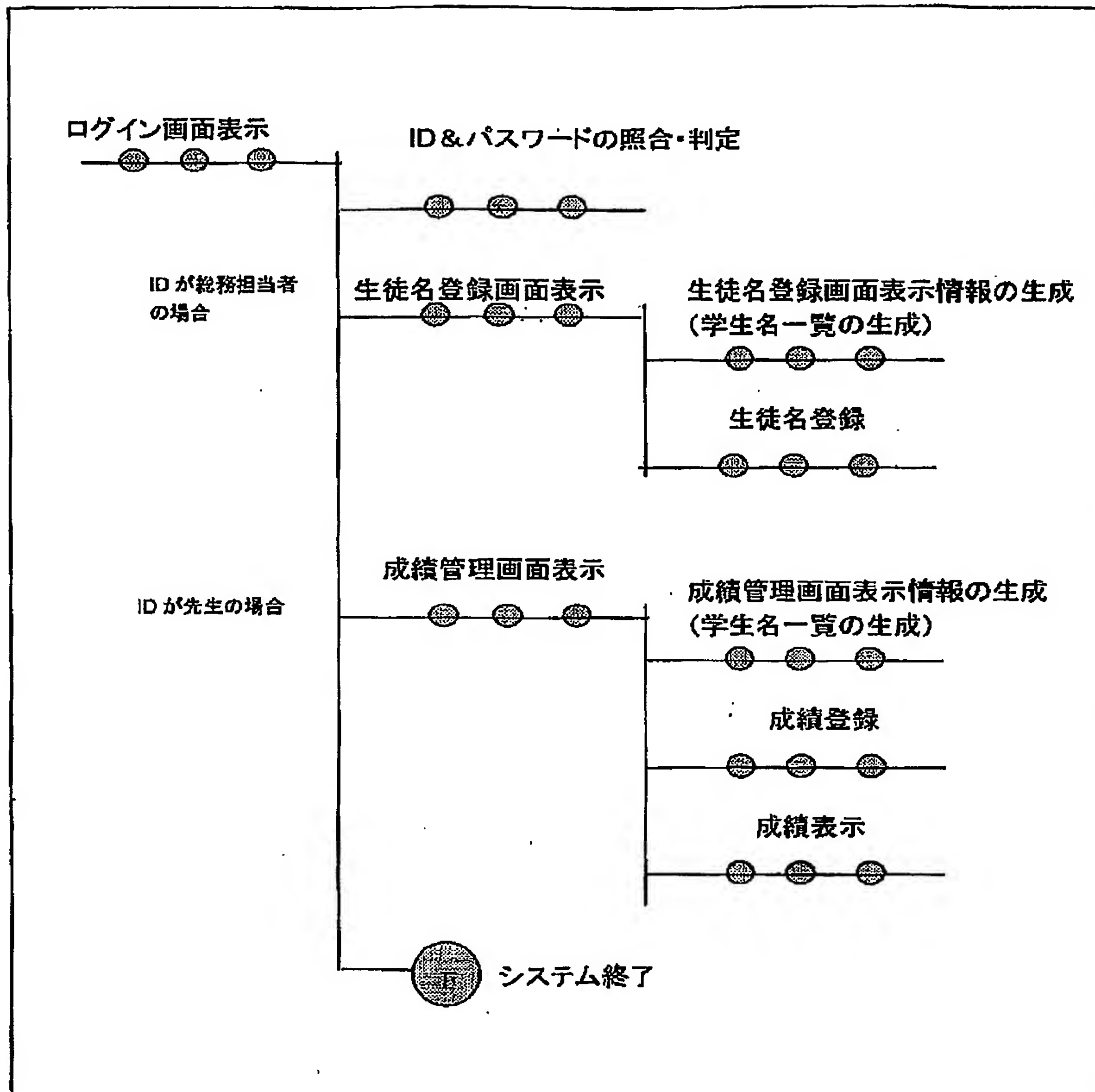


【図 1 3】



【図 1 4】

成績管理プログラムの処理経路図



【図 1.5】

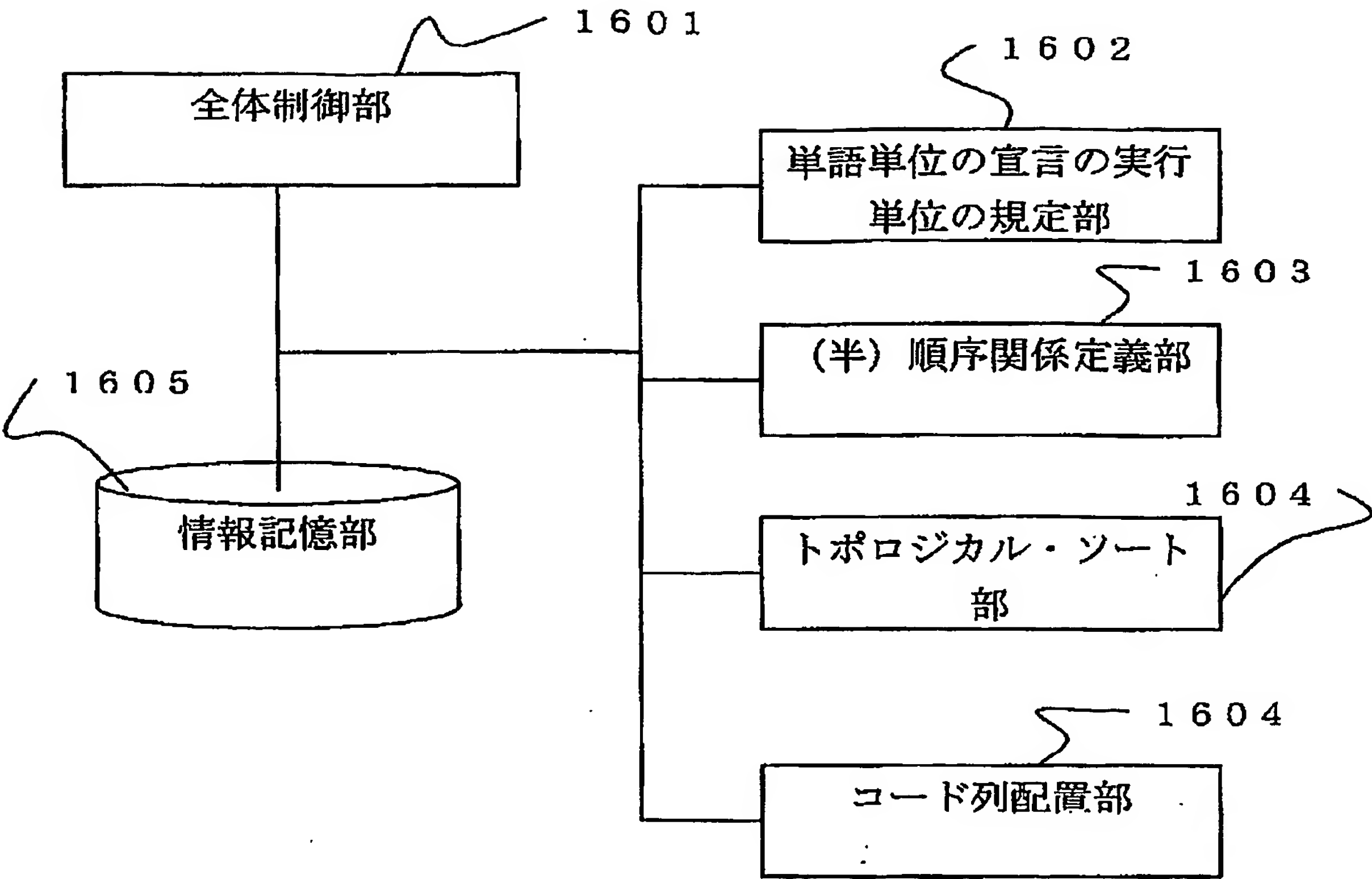
```

1  // S1
2  // S1W04
3  W04_PS_PBOXRV01_S1I_RV01_SQ;           // clear input flag
4  W04_PS_PCR1S1I_S1I_SQ;                 // clear input area
5  // S1W04 e
6  // FALSE
7  if(W02_S1I.cmdOK == FALSE)               // shared variable from route vector
8  {                                         //
9  // S1W02
10 W02_P1_PRD1RV01_S1I_RV01_SQ;           // input from screen
11 strncpy ( W02_S1I.UserID,S1_Buff.UserID, size of (W02_S1I.UserID) ); // input value
12 strncpy ( W02_S1I.Password,S1_Buff.Password, size of (W02_S1I.Password) ); // input value
13 W02_S1I.cmdOK = S1_Buff.cmdOK;           // input function button shared variable
14 W02_S1I.cmdExit = S1_Buff.cmdExit;       // input function button shared variable
15 }                                         // these word are only definition
16 // S1W02 e
17 // FALSE e
18 // Exit
19 // S1W03
20 if ( W02_S1I.cmdExit == TRUE)           // shared variable from route vector
21 {                                         //
22 // W03_PN_PNTES1W03_SQ; /* S1-W03 */ // eliminated route vector
23

```

プログラムの一部

【図 16】



【図 1 7】

